# CONTENTS

## 1 The Network Time Protocol (NTP) Distribution
1.1 The Handbook ................................................. 1
1.2 Building and Installing NTP .................................. 2
1.3 Resolving Problems ........................................... 2
1.4 Further Information ........................................... 2

## 2 Build and Install
2.1 Quick Start .................................................. 3
2.2 Building and Installing the Distribution ......................... 4
2.3 Build Options ............................................... 5
2.4 Debugging Reference Clock Drivers ............................. 5
2.5 NTP Debugging Techniques .................................. 6
2.6 Hints and Kinks .............................................. 10
2.7 NTP Bug Reporting Procedures ................................. 10

## 3 Program Manual Pages
3.1 `ntpd` - Network Time Protocol (NTP) Daemon ................. 11
3.2 `ntpq` - standard NTP query program .......................... 14
3.3 `ntpdc` - special NTP query program ......................... 22
3.4 `ntpd` - set the date and time via NTP ....................... 28
3.5 `ntpwait` - waits until ntpd is in synchronized state ....... 30
3.6 `sntp` - Simple Network Time Protocol (SNTP) Client ....... 30
3.7 `ntptrace` - trace a chain of NTP servers back to the primary source .............. 33
3.8 `tickadj` - set time-related kernel variables ................ 33
3.9 `ntptime` - read and set kernel time variables ............... 35
3.10 `ntpkeygen` - generate public and private keys ............ 35
3.11 `ntpdsim` - Network Time Protocol (NTP) simulator .......... 40
3.12 `ntpdsim` - Network Time Protocol (NTP) Simulator ......... 41

## 4 Handbook Pages
4.1 NTP Version 4 Release Notes ................................ 45
4.2 Command Index ............................................. 48
4.3 Access Control Support ...................................... 50
4.4 Association Management ...................................... 50
4.5 Authentication Support ...................................... 53
4.6 Performance Metrics ......................................... 55
4.7 Rate Management and the Kiss-o'-Death Packet ............... 58
4.8 Reference Clock Support .................................... 60

## 5 Client and Server Configuration
5.1 Automatic Server Discovery Schemes .......................... 63
10.10 Arbiter 1088A/B GPS Receiver ................................................................. 194
10.11 KSI/Odetics TPRO/S IRIG Interface ...................................................... 196
10.12 bc635VME/bc350VXI Time and Frequency Processor .......................... 197
10.13 NIST/USNO/PTB Modem Time Services ................................................ 198
10.14 Heath WWV/WWVH Receiver ................................................................. 200
10.15 Generic NMEA GPS Receiver ................................................................. 201
10.16 PPS Clock Discipline .............................................................................. 205
10.17 Hewlett Packard 58503A GPS Receiver and HP Z3801A ......................... 207
10.18 Arcron MSF Receiver .............................................................................. 208
10.19 Shared Memory Driver .......................................................................... 213
10.20 Trimble Palisade and Thunderbolt Receivers .......................................... 217
10.21 Motorola Oncore GPS receiver .............................................................. 228
10.22 Rockwell Jupiter GPS receiver ............................................................... 230
10.23 Chrono-log K-series WWVB receiver ..................................................... 231
10.24 Dumb Clock ............................................................................................ 231
10.25 Ultralink Clock ....................................................................................... 232
10.26 Conrad parallel port radio clock ............................................................... 234
10.27 Radio WWV/H Audio Demodulator/Decoder ........................................... 235
10.28 Forum Graphic GPS Dating station .......................................................... 242
10.29 hopf Serial Line Receivers (6021 and kompatible) ................................... 243
10.30 hopf PCI-Bus Receiver (6039 GPS/DCF77) ............................................. 245
10.31 JJY Receivers .......................................................................................... 246
10.32 Zyfer GPStarplus Receiver ...................................................................... 251
10.33 RIPE NCC interface for Trimble Palisade .............................................. 252
10.34 NeoClock4X - DCF77 / TDF serial line receiver ..................................... 254
10.35 Spectracom TSYNC PCI ........................................................................ 255
10.36 GPSD NG client driver ........................................................................... 255
10.37 MX4200 Receiver Data Format ............................................................... 260
10.38 Motorola ONCORE - The Shared Memory Interface .............................. 267
10.39 European Automated Computer Time Services ....................................... 271

Index 273
CHAPTER ONE

THE NETWORK TIME PROTOCOL (NTP) DISTRIBUTION

Note: The NTP Version 4 software contained in this distribution is available without charge under the conditions set forth in the Copyright Notice.

Important: It is very important that readers understand that the NTP document collection began 25 years ago and remains today a work in progress. It has evolved as new features were invented and old features retired. It has been widely copied, cached and morphed to other formats, including man pages, with varying loss of fidelity. However, these HTML pages are the ONLY authoritative and definitive reference. Readers should always use the collection that comes with the distribution they use. A copy of the online collection at www.ntp.org is normally included in the most recent snapshot, but might not agree with an earlier snapshot or release version.

This distribution is an implementation of RFC 5905 “Network Time Protocol Version 4: Protocol and Algorithms Specification”. NTP is widely used to synchronize a computer to Internet time servers or other sources, such as a radio or satellite receiver or telephone modem service. It can also be used as a server for dependent clients. It provides accuracies typically less than a millisecond on LANs and up to a few milliseconds on WANs. Typical NTP configurations utilize multiple redundant servers and diverse network paths in order to achieve high accuracy and reliability.

This distribution includes a simulation framework in which substantially all the runtime NTP operations and most features can be tested and evaluated. This has been very useful in exploring in vitro response to unusual circumstances or over time periods impractical in vivo. Details are on the Network Time Protocol (NTP) Simulator page.

1.1 The Handbook

A good deal of tutorial and directive information is available on the handbook pages. These should be read in conjunction with the command and option information available on the pages listed on the Site Map page.

NTP Version 4 Release Notes Lists recent changes and new features in the current distribution.

Association Management Describes how to configure servers and peers and manage the various options. Includes automatic server discovery schemes.

Automatic Server Discovery Schemes Describes automatic server discovery using broadcast, multicast, manycast and server pool scheme.

Access Control Support Describes the access control mechanisms that can be used to limit client access to various time and management functions.

Authentication Support Describes the authentication mechanisms for symmetric-key and public-key cryptography.

Rate Management and the Kiss-o’-Death Packet Describes the principles of rate management to minimize network load and defend against DoS attacks.
Reference Clock Support  Describes the collection of radio clocks used to synchronize primary servers.

How NTP Works  Gives an overview of the NTP daemon architecture and how it works.

1.2 Building and Installing NTP

NTP supports Unix, VMS and Windows (2000 and later) systems. The Building and Installing the Distribution page details the procedures for building and installing on a typical system. This distribution includes drivers for many radio and satellite receivers and telephone modem services in the US, Canada and Europe. A list of supported drivers is on the Reference Clock Drivers page. The default build includes the debugging options and all drivers that run on the target machine; however, options and drivers can be included or excluded using options on the Configuration Options page.

1.3 Resolving Problems

Like other things in modern Internet life, NTP problems can be devilishly intricate. This distribution includes a number of utilities designed to identify and repair problems using an integrated management protocol supported by the ntpq utility program.

The NTP Debugging Techniques and Hints and Kinks pages contain useful information for identifying problems and devising solutions. Additional information on reference clock driver construction and debugging is in the Debugging Reference Clock Drivers page.

Users are invited to report bugs and offer suggestions via the NTP Bug Reporting Procedures page.

1.4 Further Information

The Site Map page contains a list of document collections arranged by topic. The Program Manual Pages collection may be the best place to start. The Command Index collection contains a list of all configuration file commands together with a short function description. A great wealth of additional information is available via the External Links collection, including a book and numerous background papers and briefing presentations.

Background information on computer network time synchronization is on the Executive Summary - Computer Network Time Synchronization page. Discussion on new features and interoperability with previous NTP versions is on the NTP Version 4 Release Notes page. Background information, bibliography and briefing slides suitable for presentations are on the Network Time Synchronization Research Project page. Additional information is at the NTP website www.ntp.org.
CHAPTER TWO

BUILD AND INSTALL

2.1 Quick Start

For the rank amateur the sheer volume of the documentation collection must be intimidating. However, it doesn’t take much to fly the ntpd daemon with a simple configuration where a workstation needs to synchronize to some server elsewhere in the Internet. The first thing is to build the distribution for the particular workstation and install in the usual place. The Building and Installing the Distribution page describes how to do this.

While it is possible that certain configurations do not need a configuration file, most do. The file, called by default /etc/ntp.conf, need only contain one command specifying a remote server, for instance

```
server foo.bar.com
```

Choosing an appropriate remote server is somewhat of a black art, but a suboptimal choice is seldom a problem. The simplest and best is to use the Server Pool Scheme on the Automatic Server Discovery page. There are about two dozen public time servers operated by the National Institutes of Science and Technology (NIST), US Naval Observatory (USNO), Canadian National Research Council (NRC) and many others available on the Internet. Lists of public primary and secondary NTP servers maintained on the Public NTP Time Servers page, which is updated frequently. The lists are sorted by country and, in the case of the US, by state. Usually, the best choice is the nearest in geographical terms, but the terms of engagement specified in each list entry should be carefully respected.

During operation ntpd measures and corrects for incidental clock frequency error and occasionally writes the current value to a file specified by the

```
driftfile /etc/ntp.drift
```

configuration command. If ntpd is stopped and restarted, it initializes the frequency from this file and avoids the potentially lengthy interval to relearn the correction.

That’s all there is to it, unless some problem in network connectivity or local operating system configuration occurs. The most common problem is some firewall between the workstation and server. System administrators should understand NTP uses UDP port 123 as both the source and destination port and that NTP does not involve any operating system interaction other than to set the system clock. While almost all modern Unix systems have included NTP and UDP port 123 defined in the services file, this should be checked if ntpd fails to come up at all.

The best way to confirm NTP is working is using the ntpq utility, although the ntpdc utility may be useful in extreme cases. See the documentation pages for further information. Don’t forget to check for system log messages. In the most extreme cases the –d option on the ntpd command line results in a blow-by-blow trace of the daemon operations. While the trace output can be cryptic, to say the least, it gives a general idea of what the program is doing and, in particular, details the arriving and departing packets and any errors found.
2.2 Building and Installing the Distribution

It is not possible in a software distribution such as this to support every individual computer and operating system with a common executable, even with the same system but different versions and options. Therefore, it is necessary to configure, build and install for each system and version. In almost all cases, these procedures are completely automatic. The user types .configure, and make install in that order and the autoconfigure system does the rest. There are some exceptions, as noted below and on the Hints and Kinks page.

If available, the OpenSSL library from http://www.openssl.org is used to support public key cryptography. The library must be built and installed prior to building NTP. The procedures for doing that are included in the OpenSSL documentation. The library is found during the normal NTP configure phase and the interface routines compiled automatically. Only the libcrypto.a library file and openssl header files are needed. If the library is not available or disabled, this step is not required.

The Build Options page describes a number of options that determine whether debug support is included, whether and which reference clock drivers are included and the locations of the executables and library files, if not the default. By default debugging options and all reference clock drivers are included.

2.2.1 Building and Installing for Unix

This distribution uses common compilers and tools that come with most Unix distributions. Not all of these tools exist in the standard distribution of modern Unix versions (compilers are likely to be an add-on product). If this is the case, consider using the GNU tools and gcc compiler included as freeware in some systems. For a successful build, all of these tools should be accessible via the current path.

The first thing to do is uncompress the distribution and extract the source tree. In the distribution base directory use the .configure command to perform an automatic configuration procedure. This command inspects the hardware and software environment and configures the build process accordingly. Use the make command to compile and link the distribution and the install command to install the executables by default in /usr/local/bin.

If your site supports multiple architectures and uses NFS to share files, you can use a single source tree to build executables for multiple architectures. While running on a particular architecture, change to the base directory and create a subdirectory using a command like mkdir A.machine, which will create an architecture-specific directory, then change to this directory and mumble ../configure. The remaining steps are the same whether building in the base directory or in the subdirectory.

2.2.2 Building and Installing for Windows

NTP supports Windows 2000 and later. See the NTP 4.x for Windows NT page for directions to compile the sources and install the executables. A precompiled executable is available.

2.2.3 Configuration

You are now ready to configure the daemon. You will need to create a NTP configuration file by default in /etc/ntp.conf. Newbies should see the Quick Start page for orientation. Seasoned veterans can start with the ntpd - Network Time Protocol (NTP) daemon page and move on to the specific configuration option pages from there.

2.2.4 If You Have Problems

If you have problems with your hardware and software environment (e.g. operating system-specific issues), browse the Hints and Kinks pages. For other problems a tutorial on debugging technique is in the NTP Debugging Techniques page. A list of important system log messages is on the ntpd System Log Messages page.
The first line of general assistance is the NTP web site www.ntp.org and the helpful documents resident there. Requests for assistance of a general nature and of interest to other timekeepers should be sent to the NTP newsgroup comp.protocols.time.ntp.

Users are invited to report bugs and offer suggestions via the NTP Bug Reporting Procedures page.

### 2.2.5 Additional make commands

**make clean**

Cleans out object files, programs and temporary files.

**make distclean**

Does the work of clean, but cleans out all directories in preparation for a new distribution release.

**make dist**

Does the work of make distclean, but constructs compressed tar files for distribution. You must have GNU automake to perform this function.

### 2.3 Build Options

Most modern software distributions include an autoconfigure utility which customizes the build and install configuration according to the specific hardware, operating system and file system conventions. For NTP this utility is called configure, which is run before building and installing the program components. For most installations no additional actions are required other than running configure with no options. However, it is possible to customize the build and install configuration through the use of configure options.

The available options, together with a concise description, can be displayed by running configure with the --help option. Various options can be used to reduce the memory footprint, adjust the scheduling priority, enable or disable debugging support or reference clock driver support. The options can be used to specify where to install the program components or where to find various libraries if they are not in the default place.

### 2.4 Debugging Reference Clock Drivers

The ntpq and ntpdc utility programs can be used to debug reference clocks, either on the server itself or from another machine elsewhere in the network. The server is compiled, installed and started using the configuration file described in the ntpd page and its dependencies. If the clock appears in the ntpq utility and pe command, no errors have occurred and the daemon has started, opened the devices specified and waiting for peers and radios to come up. If not, the first thing to look for are error messages on the system log. These are usually due to improper configuration, missing links or multiple instances of the daemon.

It normally takes a minute or so for evidence to appear that the clock is running and the driver is operating correctly. The first indication is a nonzero value in the reach column in the pe billboard. If nothing appears after a few minutes, the next step is to be sure the RS232 messages, if used, are getting to and from the clock. The most reliable way to do this is with an RS232 tester and to look for data flashes as the driver polls the clock and/or as data arrive from the clock. Our experience is that the overwhelming fraction of problems occurring during installation are due to problems such as miswired connectors or improperly configured device links at this stage.

If RS232 messages are getting to and from the clock, the variables of interest can be inspected using the ntpq program and various commands described on the documentation page. First, use the pe and as commands to display billboards showing the peer configuration and association IDs for all peers, including the radio clock. The assigned clock address should appear in the pe billboard and the association ID for it at the same relative line position in the as billboard.
Additional information is available with the `rv` and `clockvar` commands, which take as argument the association ID shown in the `as` billboard. The `rv` command with no argument shows the system variables, while the `rv` command with association ID argument shows the peer variables for the clock, as well as other peers of interest. The `clockvar` command with argument shows the peer variables specific to reference clock peers, including the clock status, device name, last received timecode (if relevant), and various event counters. In addition, a subset of the `fudge` parameters is included. The poll and error counters in the `clockvar` billboard are useful debugging aids. The `poll` counts the poll messages sent to the clock, while the `noreply`, `badformat` and `baddate` count various errors. Check the timecode to be sure it matches what the driver expects. This may require consulting the clock hardware reference manual, which is probably pretty dusty at this stage.

The `ntpd` utility program can be used for detailed inspection of the clock driver status. The most useful are the `clockstat` and `clkbug` commands described in the document page. While these commands permit getting quite personal with the particular driver involved, their use is seldom necessary, unless an implementation bug shows up. If all else fails, turn on the debugging trace using two `-d` flags in the `ntpd` startup command line. Most drivers will dump status at every received message in this case. While the displayed trace can be intimidating, this provides the most detailed and revealing indicator of how the driver and clock are performing and where bugs might lurk.

Most drivers write a message to the `clockstats` file as each timecode or surrogate is received from the radio clock. By convention, this is the last ASCII timecode (or ASCII gloss of a binary-coded one) received from the radio clock. This file is managed by the `filegen` facility described in the `ntpd` page and requires specific commands in the configuration file. This forms a highly useful record to discover anomalies during regular operation of the clock. The scripts included in the `./scripts/stats` directory can be run from a `cron` job to collect and summarize these data on a daily or weekly basis. The summary files have proven inspirational to detect infrequent misbehavior due to clock implementation bugs in some radios.

### 2.5 NTP Debugging Techniques

#### 2.5.1 Initial Startup

This page discusses `ntpd` program monitoring and debugging techniques using the `ntpq` - standard NTP query program, either on the local server or from a remote machine. In special circumstances the `ntpd` - special NTP query program, can be useful, but its use is not covered here. The `ntpq` program implements the management functions specified in the NTP specification RFC 1305, Appendix A. It is used to read and write the variables defined in the NTP Version 4 specification now navigating the standards process. In addition, the program can be used to send remote configuration commands to the server.

The `ntpd` daemon can operate in two modes, depending on the presence of the `-d` command-line option. Without the option the daemon detaches from the controlling terminal and proceeds autonomously. With one or more `-d` options the daemon does not detach and generates special trace output useful for debugging. In general, interpretation of this output requires reference to the sources. However, a single `-d` does produce only mildly cryptic output and can be very useful in finding problems with configuration and network troubles.

Some problems are immediately apparent when the daemon first starts running. The most common of these are the lack of a UDP port for NTP (123) in the Unix `/etc/services` file (or equivalent in some systems). **Note that NTP does not use TCP in any form. Also note that NTP requires port 123 for both source and destination ports.** These facts should be pointed out to firewall administrators.

Other problems are apparent in the system log, which ordinarily shows the startup banner, some cryptic initialization data and the computed precision value. Event messages at startup and during regular operation are sent to the optional `protostats` monitor file, as described on the `Event Messages and Status Words` page. These and other error messages are sent to the system log, as described on the `ntpd System Log Messages` page. In real emergencies the daemon will send a terminal error message to the system log and then cease operation.

The next most common problem is incorrect DNS names. Check that each DNS name used in the configuration file exists and that the address responds to the Unix `ping` command. The Unix `traceroute` or Windows `tracert`
utility can be used to verify a partial or complete path exists. Most problems reported to the NTP newsgroup are not NTP problems, but problems with the network or firewall configuration.

### 2.5.2 Verifying Correct Operation

Unless using the `iburst` option, the client normally takes a few minutes to synchronize to a server. If the client time at startup happens to be more than 1000 s distant from NTP time, the daemon exits with a message to the system log directing the operator to manually set the time within 1000 s and restart. If the time is less than 1000 s but more than 128 s distant, a step correction occurs and the daemon restarts automatically.

When started for the first time and a frequency file is not present, the daemon enters a special mode in order to calibrate the frequency. This takes 900 s during which the time is not disciplined. When calibration is complete, the daemon creates the frequency file and enters normal mode to amortize whatever residual offset remains.

The `ntpq` commands `pe`, `as` and `rv` are normally sufficient to verify correct operation and assess nominal performance. The `pe` command displays a list showing the DNS name or IP address for each association along with selected status and statistics variables. The first character in each line is the tally code, which shows which associations are candidates to set the system clock and of these which one is the system peer. The encoding is shown in the `select` field of the `peer status word`.

The `as` command displays a list of associations and association identifiers. Note the `condition` column, which reflects the tally code. The `rv` command displays the `system variables` billboard, including the `system status word`. The `rv <assocID>` command, where `<assocID>` is the association ID, displays the `peer variables` billboard, including the `peer status word`. Note that, except for explicit calendar dates, times are in milliseconds and frequencies are in parts-per-million (PPM).

A detailed explanation of the system, peer and clock variables in the billboards is beyond the scope of this page; however, a comprehensive explanation for each one is in the NTPv4 protocol specification. The following observations will be useful in debugging and monitoring.

1. The server has successfully synchronized to its sources if the `leap` peer variable has value other than 3 (11b). The client has successfully synchronized to the server when the `leap` system variable has value other than 3.

2. The `reach` peer variable is an 8-bit shift register displayed in octal format. When a valid packet is received, the rightmost bit is lit. When a packet is sent, the register is shifted left one bit with 0 replacing the rightmost bit. If the `reach` value is nonzero, the server is reachable; otherwise, it is unreachable. Note that, even if all servers become unreachable, the system continues to show valid time to dependent applications.

3. A useful indicator of miscellaneous problems is the `flash` peer variable, which shows the result of 13 sanity tests. It contains the `flash status word` bits, commonly called flashers, which displays the current errors for the association. These bits should all be zero for a valid server.

4. The three peer variables `filtdelay`, `filtoffset` and `filtdisp` show the delay, offset and jitter statistics for each of the last eight measurement rounds. These statistics and their trends are valuable performance indicators for the server, client and the network. For instance, large fluctuations in delay and jitter suggest network congestion. Missing clock filter stages suggest packet losses in the network.

5. The synchronization distance, defined as one-half the delay plus the dispersion, represents the maximum error statistic. The jitter represents the expected error statistic. The maximum error and expected error calculated from the peer variables represents the quality metric for the server. The maximum error and expected error calculated from the system variables represents the quality metric for the client. If the root synchronization distance for any server exceeds 1.5 s, called the select threshold, the server is considered invalid.

### 2.5.3 Large Frequency Errors

The frequency tolerance of computer clock oscillators varies widely, sometimes above 500 PPM. While the daemon can handle frequency errors up to 500 PPM, or 43 seconds per day, values much above 100 PPM reduce the head-
room, especially at the lowest poll intervals. To determine the particular oscillator frequency, start ntpd using the noselect option with the server configuration command.

Record the time of day and offset displayed by the ntpq pe command. Wait for an hour or so and record the time of day and offset. Calculate the frequency as the offset difference divided by the time difference. If the frequency is much above 100 PPM, the tickadj program might be useful to adjust the kernel clock frequency below that value. For systems that do not support this program, this might be one using a command in the system startup file.

### 2.5.4 Access Controls

Provisions are included in ntpd for access controls which deflect unwanted traffic from selected hosts or networks. The controls described on the Access Control Options include detailed packet filter operations based on source address and address mask. Normally, filtered packets are dropped without notice other than to increment tally counters. However, the server can be configured to send a “kiss-o’-death” (KoD) packet to the client either when explicitly configured or when cryptographic authentication fails for some reason. The client association is permanently disabled, the access denied bit (TEST4) is set in the flash variable and a message is sent to the system log.

The access control provisions include a limit on the packet rate from a host or network. If an incoming packet exceeds the limit, it is dropped and a KoD sent to the source. If this occurs after the client association has synchronized, the association is not disabled, but a message is sent to the system log. See the Access Control Options page for further information.

### 2.5.5 Large Delay Variations

In some reported scenarios an access line may show low to moderate network delays during some period of the day and moderate to high delays during other periods. Often the delay on one direction of transmission dominates, which can result in large time offset errors, sometimes in the range up to a few seconds. It is not usually convenient to run ntpd throughout the day in such scenarios, since this could result in several time steps, especially if the condition persists for greater than the stepout threshold.

Specific provisions have been built into ntpd to cope with these problems. The scheme is called “huff-'n-puff and is described on the Miscellaneous Options page. An alternative approach in such scenarios is first to calibrate the local clock frequency error by running ntpd in continuous mode during the quiet interval and let it write the frequency to the ntp.drift file. Then, run ntpd -q from a cron job each day at some time in the quiet interval. In systems with the nanokernel or microkernel performance enhancements, including Solaris, Tru64, Linux and FreeBSD, the kernel continuously disciplines the frequency so that the residual correction produced by ntpd is usually less than a few milliseconds.

### 2.5.6 Cryptographic Authentication

Reliable source authentication requires the use of symmetric key or public key cryptography, as described on the Authentication Options page. In symmetric key cryptography servers and clients share session keys contained in a secret key file In public key cryptography, which requires the OpenSSL software library, the server has a private key, never shared, and a public key with unrestricted distribution. The cryptographic media required are produced by the ntp-keygen program.

Problems with symmetric key authentication are usually due to mismatched keys or improper use of the trustedkey command. A simple way to check for problems is to use the trace facility, which is enabled using the ntpd -d command line. As each packet is received a trace line is displayed which shows the authentication status in the auth field. A status of 1 indicates the packet was successful authenticated; otherwise it has failed.

A common misconception is the implication of the auth bit in the enable and disable commands. This bit does not affect authentication in any way other than to enable or disable mobilization of a new persistent association in broadcast/multicast client, manycast client or symmetric passive modes. If enabled, which is the default, these
associations require authentication; if not, an association is mobilized even if not authenticated. Users are cautioned that running with authentication disabled is very dangerous, since an intruder can easily strike up an association and inject false time values.

Public key cryptography is supported in NTPv4 using the Autokey protocol, which is described in briefings on the NTP Project page linked from www.ntp.org. Development of this protocol is mature and the ntpd implementation is basically complete. Autokey version 2, which is the latest and current version, includes provisions to hike certificate trails, operate as certificate authorities and verify identity using challenge/response identification schemes. Further details of the protocol are on the Authentication Options page. Common problems with configuration and key generation are mismatched key files, broken links and missing or broken random seed file.

As in the symmetric key cryptography case, the trace facility is a good way to verify correct operation. A statistics file cryptostats records protocol transactions and error messages. The daemon requires a random seed file, public/private key file and a valid certificate file; otherwise it exits immediately with a message to the system log. As each file is loaded a trace message appears with its filestamp. There are a number of checks to insure that only consistent data are used and that the certificate is valid. When the protocol is in operation a number of checks are done to verify the server has the expected credentials and its filestamps and timestamps are consistent. Errors found are reported using NTP control and monitoring protocol traps with extended trap codes shown in the Authentication Options page.

To assist debugging every NTP extension field is displayed in the trace along with the Autokey operation code. Every extension field carrying a verified signature is identified and displayed along with filestamp and timestamp where meaningful. In all except broadcast/multicast client mode, correct operation of the protocol is confirmed by the absence of extension fields and an auth value of one. It is normal in broadcast/multicast client mode that the broadcast server use one extension field to show the host name, status word and association ID.

### 2.5.7 Debugging Checklist

If the ntpq or ntpdc programs do not show that messages are being received by the daemon or that received messages do not result in correct synchronization, verify the following:

1. Verify the /etc/services file host machine is configured to accept UDP packets on the NTP port 123. NTP is specifically designed to use UDP and does not respond to TCP.
2. Check the system log for ntpd messages about configuration errors, name-lookup failures or initialization problems. Common system log messages are summarized on the ntpd System Log Messages page. Check to be sure that only one copy of ntpd is running.
3. Verify using ping or other utility that packets actually do make the round trip between the client and server. Verify using nslookup or other utility that the DNS server names do exist and resolve to valid Internet addresses.
4. Check that the remote NTP server is up and running. The usual evidence that it is not is a Connection refused message.
5. Using the ntpdc program, verify that the packets received and packets sent counters are incrementing. If the sent counter does not increment and the configuration file includes configured servers, something may be wrong in the host network or interface configuration. If this counter does increment, but the received counter does not increment, something may be wrong in the network or the server NTP daemon may not be running or the server itself may be down or not responding.
6. If both the sent and received counters do increment, but the reach values in the pe billboard with ntpq continues to show zero, received packets are probably being discarded for some reason. If this is the case, the cause should be evident from the flash variable as discussed above and on the ntpq page. It could be that the server has disabled access for the client address, in which case the refid field in the ntpq pe billboard will show a kiss code. See earlier on this page for a list of kiss codes and their meaning.
7. If the `reach` values in the `pe` billboard show the servers are alive and responding, note the tattletale symbols at the left margin, which indicate the status of each server resulting from the various grooming and mitigation algorithms. The interpretation of these symbols is discussed on the `ntpq` page. After a few minutes of operation, one or another of the reachable server candidates should show a * tattletale symbol. If this doesn’t happen, the intersection algorithm, which classifies the servers as truechimers or falsetickers, may be unable to find a majority of truechimers among the server population.

8. If all else fails, see the FAQ and/or the discussion and briefings at the NTP Project page.

2.6 Hints and Kinks

This is an index for a set of troubleshooting notes contained in individual text files in the `./hints` directory. They were supplied by various volunteers in the form of mail messages, patches or just plain word of mouth. Each note applies to a specific computer and operating system and gives information found useful in setting up the NTP distribution or site configuration. The notes are very informal and subject to errors; no attempt has been made to verify the accuracy of the information contained in them.

Additions or corrections to this list or the information contained in the notes is solicited. The most useful submissions include the name of the computer manufacturer (and model numbers where appropriate), operating system (specific version(s) where appropriate), problem description, problem solution and submitter’s name and electric address. If the submitter is willing to continue debate on the problem, please so advise. See the directory listing.

2.7 NTP Bug Reporting Procedures

2.7.1 Security Bug Reporting Procedures

If you find or suspect a security related program bug in this distribution, please send a report to security@ntp.org. Please do not contact developers directly.

2.7.2 Non-Security Bug Reporting Procedures

If you find or suspect a non-security related program or documentation bug in this distribution, please send a report to the NTP Public Service Project Bug Tracking System (Bugzilla) at http://bugs.ntp.org/. Bugs reported this way are immediately forwarded to the developers. Please do not contact the developers directly.

If you wish to send a report via electronic mail, please remember that your report will be held until one of our volunteers enters it in Bugzilla. The email address for these reports is bugs@ntp.org. You will need to register at http://bugs.ntp.org/ to participate directly in any e-mail discussion regarding your report. If you don’t register and we have questions for you we won’t be able to make progress on fixing your problem. Please directly register on and use our Bugzilla instance to report issues.
3.1 _ntpd_ - Network Time Protocol (NTP) Daemon

### 3.1.1 Synopsis

```
ntpd [ -46aAbdDgLnNqx ] [ -c conffile ] [ -f driftfile ] [ -i jaildir ] [ -I InterfaceOrAddress ] [ -k keyfile ] [ -l logfile ] [ -p pidfile ] [ -P priority ] [ -r broadcastdelay ] [ -s statsdir ] [ -t key ] [ -u user[:group] ] [ -U interface_update_interval ] [ -v variable ] [ -V variable ]
```

### 3.1.2 Description

The _ntpd_ program is an operating system daemon that synchronizes the system clock to remote NTP time servers or local reference clocks. It is a complete implementation of NTP version 4 defined by RFC 5905, but also retains compatible with version 3 defined by RFC 1305 and versions 1 and 2, defined by RFC 1059 and RFC 1119, respectively.

The program can operate in any of several modes, including client/server, symmetric and broadcast modes, and with both symmetric-key and public key-cryptography.

The _ntpd_ program ordinarily requires a configuration file described on this page. It contains configuration commands described on the pages listed above. However a client can discover remote servers and configure them automatically. This makes it possible to deploy a fleet of workstations without specifying configuration details specific to the local environment. Further details are on the

The _ntpd_ program normally operates continuously while adjusting the system time and frequency, but in some cases this might not be practical. With the `-q` option _ntpd_ operates as in continuous mode, but exits just after setting the clock for the first time. Most applications will probably want to specify the `iburst` option with the `server` command. With this option a volley of messages is exchanged to groom the data and set the clock in about ten seconds. If nothing is heard after a few minutes, the daemon times out and exits without setting the clock.

### 3.1.3 Command Line Options

- `-4`
  
  Force DNS resolution of host names to the IPv4 namespace.

- `-6`
  
  Force DNS resolution of host names to the IPv6 namespace.

- `-a`
  
  Require cryptographic authentication for broadcast client, multicast client and symmetric passive associations. This is the same operation as the `enable auth` command and is the default.
-A
Do not require cryptographic authentication for broadcast client, multicast client and symmetric passive associations. This is the same operation as the disable auth command and almost never a good idea.

-b
Enable the client to synchronize to broadcast servers.

-c <conffile>
Specify the name and path of the configuration file. Without the option the default is /etc/ntp.conf.

-d
Disable switching into daemon mode, so ntpd stays attached to the starting terminal which will get all the debugging printout. Also, ^C will kill it. This option may occur more than once, with each occurrence indicating greater detail of display.

-D <level>
Specify debugging level directly, with level corresponding to the number of -d options.

-f <driftfile>
Specify the name and path of the frequency file. This is the same operation as the driftfile driftfile configuration command.

-g
Normally, ntpd exits with a message to the system log if the offset exceeds the panic threshold, which is 1000 s by default. This option allows the time to be set to any value without restriction; however, this can happen only once. If the threshold is exceeded after that, ntpd will exit with a message to the system log. This option can be used with the -q and -x options. See the tinker command for other options.

-i <jaildir>
Chroot the server to the directory jaildir. This option also implies that the server attempts to drop root privileges at startup (otherwise, chroot gives very little additional security), and it is only available if the OS supports to run the server without full root privileges. You may need to also specify a -u option.

-I [address | interface name]
Open the network address given, or all the addresses associated with the given interface name. This option may appear multiple times. This option also implies not opening other addresses, except wildcard and localhost. This option is deprecated. Please consider using the configuration file interface command, which is more versatile.

-k <keyfile>
Specify the name and path of the symmetric key file. This is the same operation as the keys keyfile command.

-l <logfile>
Specify the name and path of the log file. The default is the system log file. This is the same operation as the logfile logfile command.

-m
Once the system clock is synchronized, register with mDNS as an available server.

-L
Do not listen to virtual interfaces, defined as those with names containing a colon. This option is deprecated. Please consider using the configuration file interface command, which is more versatile.

-M
Raise scheduler precision to its maximum (1 ms) using timeBeginPeriod. (Windows only)

-n
Don’t fork.

-N
To the extent permitted by the operating system, run the ntpd at the highest priority.
-p <pidfile>
Specify the name and path of the file used to record the ntpd process ID. This is the same operation as the pidfile pidfile command.

-P <priority>
To the extent permitted by the operating system, run the ntpd at the specified priority.

-q
Exit the ntpd just after the first time the clock is set. This behavior mimics that of the ntpdate program, which is to be retired. The -g and -x options can be used with this option. Note: The kernel time discipline is disabled with this option.

-r <broadcastdelay>
Specify the default propagation delay from the broadcast/multicast server to this client. This is necessary only if the delay cannot be computed automatically by the protocol.

-s <statsdir>
Specify the directory path for files created by the statistics facility. This is the same operation as the statsdir statsdir command.

-t <key>
Add a key number to the trusted key list. This option can occur more than once. This is the same operation as the trustedkey key command.

-u user[:group]
Specify a user, and optionally a group, to switch to. This option is only available if the OS supports running the server without full root privileges. Currently, this option is supported under NetBSD (configure with --enable-clockctl) and Linux (configure with --enable-linuxcaps).

-U number, --updateinterval=number
Number of seconds to wait between interface list scans to pick up old and delete network interface. Set to 0 to disable dynamic interface list updating. The default is to scan every 5 minutes.

-v <variable>
-V <variable>
Add a system variable listed by default.

-x
Normally, the time is slewed if the offset is less than the step threshold, which is 128 ms by default, and stepped if above the threshold. This option sets the threshold to 600 s, which is well within the accuracy window to set the clock manually. Note: Since the slew rate of typical Unix kernels is limited to 0.5 ms/s, each second of adjustment requires an amortization interval of 2000 s. Thus, an adjustment as much as 600 s will take almost 14 days to complete. This option can be used with the -g and -q options. See the tinker command for other options. Note: The kernel time discipline is disabled with this option.

--pccfreq <frequency>
Substitute processor cycle counter for QueryPerformanceCounter unconditionally using the given frequency (in Hz). --pccfreq can be used on systems which do not use the PCC to implement QueryPerformanceCounter and have a fixed PCC frequency. The frequency specified must be accurate within 0.5 percent. --usepcc is equivalent on many systems and should be tried first, as it does not require determining the frequency of the processor cycle counter. For x86-compatible processors, the PCC is also referred to as RDTSC, which is the assembly-language instruction to retrieve the current value. (Windows only)

--usepcc
Substitute processor cycle counter for QueryPerformanceCounter if they appear equivalent. This option should be used only if the PCC frequency is fixed. Power-saving functionality on many laptops varies the PCC frequency. (Windows only)
3.1.4 The Configuration File

Ordinarily, ntpd reads the ntp.conf configuration file at startup in order to determine the synchronization sources and operating modes. It is also possible to specify a working, although limited, configuration entirely on the command line, obviating the need for a configuration file. This may be particularly useful when the local host is to be configured as a broadcast client, with servers determined by listening to broadcasts at run time.

Usually, the configuration file is installed as /etc/ntp.conf, but could be installed elsewhere (see the -c conffile command line option). The file format is similar to other Unix configuration files - comments begin with a # character and extend to the end of the line; blank lines are ignored.

Configuration commands consist of an initial command keyword followed by a list of option keywords separated by whitespace. Commands may not be continued over multiple lines. Options may be host names, host addresses written in numeric, dotted-quad form, integers, floating point numbers (when specifying times in seconds) and text strings. Optional arguments are delimited by [ ] in the options pages, while alternatives are separated by |. The notation [ ... ] means an optional, indefinite repetition of the last item before the [ ... ].

3.1.5 Files

<table>
<thead>
<tr>
<th>File</th>
<th>Default</th>
<th>Option</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration file</td>
<td>/etc/ntp.conf</td>
<td>-c</td>
<td>conffile</td>
</tr>
<tr>
<td>frequency file</td>
<td>none</td>
<td>-f</td>
<td>driftfile</td>
</tr>
<tr>
<td>leapseconds file</td>
<td>none</td>
<td></td>
<td>leapfile</td>
</tr>
<tr>
<td>process ID file</td>
<td>none</td>
<td>-p</td>
<td>pidfile</td>
</tr>
<tr>
<td>log file</td>
<td>system log</td>
<td>-l</td>
<td>logfile</td>
</tr>
<tr>
<td>include file</td>
<td>none</td>
<td>none</td>
<td>includefile</td>
</tr>
<tr>
<td>statistics path</td>
<td>/var/NTP</td>
<td>-s</td>
<td>statsdir</td>
</tr>
<tr>
<td>keys path</td>
<td>/usr/local/etc</td>
<td>none</td>
<td>keysdir</td>
</tr>
</tbody>
</table>

3.2 ntpq - standard NTP query program

3.2.1 Synopsis

ntpq [-46dinp] [-c command] [host] [...]

3.2.2 Description

The ntpq utility program is used to monitor NTP daemon ntpd operations and determine performance. It uses the standard NTP mode 6 control message formats defined in Appendix B of the NTPv3 specification RFC 1305. The same formats are used in NTPv4, although some of the variable names have changed and new ones added. The description on this page is for the NTPv4 variables.

The program can be run either in interactive mode or controlled using command line arguments. Requests to read and write arbitrary variables can be assembled, with raw and pretty-printed output options being available. The ntpq can also obtain and print a list of peers in a common format by sending multiple queries to the server.

If one or more request options is included on the command line when ntpq is executed, each of the requests will be 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 will attempt to read commands from the standard input and execute these 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 will prompt for commands if the standard input is a terminal device.
ntpq uses NTP mode 6 packets to communicate with the NTP server, and hence can be used to query any compatible server on the network which permits it. Note that since NTP is a UDP protocol this communication will be somewhat unreliable, especially over large distances in terms of network topology. ntpq makes one attempt to retransmit requests, and will time requests out if the remote host is not heard from within a suitable timeout time.

Note that in contexts where a host name is expected, a -4 qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a -6 qualifier forces DNS resolution to the IPv6 namespace.

For examples and usage, see the NTP Debugging Techniques page.

Command line options are described following. Specifying a command line option other than -i or -n will cause the specified query (queries) to be sent to the indicated host(s) immediately. Otherwise, ntpq will attempt to read interactive format commands from the standard input.

-4
Force DNS resolution of following host names on the command line to the IPv4 namespace.

-6
Force DNS resolution of following host names on the command line to the IPv6 namespace.

-c
The following argument is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). Multiple -c options may be given.

-d
Turn on debugging mode.

-i
Force ntpq to operate in interactive mode. Prompts will be 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.

3.2.3 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, but optionally the output of individual commands may 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. These are described following.

**help** [command_keyword]
A ? by itself will print a list of all the command keywords known to ntpq. A ? followed by a command keyword will print function and usage information about the command.

**addvars** name [ = value] [...]
**rmvars** name [...] 
**clearvars**
The arguments to this command consist of a list of items of the form name = value, where the = value is ignored, and can be omitted in read requests. 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 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
can be used to remove individual variables from the list, while the `clearlist` command removes all variables from the list.

cooked
Display server messages in prettyprint format.

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

delay <milliseconds>
Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (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 <name>
Set the host to which future queries will be sent. The name may be either a DNS name or a numeric address.

hostnames [yes | no]
If `yes` is specified, host names are printed in information displays. If `no` is specified, numeric addresses are printed instead. The default is `yes`, unless modified using the command line `-n` switch.

keyid <keyid>
This command specifies the key number to be used to authenticate configuration requests. This must correspond to a key ID configured in `ntp.conf` for this purpose.

keytype
Specify the digest algorithm to use for authenticated requests, with default MD5. If the OpenSSL library is installed, digest can be be any message digest algorithm supported by the library. The current selections are: MD2, MD4, MD5, MDC2, RIPEMD160, SHA and SHA1.

ntpversion 1 | 2 | 3 | 4
Sets the NTP version number which `ntpq` claims in packets. Defaults to 2. Note that mode-6 control messages (and modes, for that matter) didn't exist in NTP version 1.

passwd
This command prompts for a password to authenticate requests. The password must correspond to the key ID configured in `ntp.conf` for this purpose.

quit
Exit `ntpq`.

raw
Display server messages as received and without reformatting.

timeout <milliseconds>
Specify a timeout period for responses to server queries. The default is about 5000 milliseconds. Note that since `ntpq` retries each query once after a timeout, the total waiting time for a timeout will be twice the timeout value set.

3.2.4 Control Message Commands

Association IDs are used to identify system, peer and clock variables. System variables are assigned an association ID of zero and system name space, while each association is assigned a nonzero association ID and peer namespace. Most control commands send a single mode-6 message to the server and expect a single response message. The exceptions are the `peers` command, which sends a series of messages, and the `mreadlist` and `mreadvar` commands, which iterate over a range of associations.

associations
Display a list of mobilized associations in the form
ind assid status conf reach auth condition last_event cnt

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind</td>
<td>index on this list</td>
</tr>
<tr>
<td>assid</td>
<td>association ID</td>
</tr>
<tr>
<td>status</td>
<td>peer status word</td>
</tr>
<tr>
<td>conf</td>
<td>yes: persistent, no: ephemeral</td>
</tr>
<tr>
<td>reach</td>
<td>yes: reachable, no: unreachable</td>
</tr>
<tr>
<td>auth</td>
<td>ok, yes, bad and none</td>
</tr>
<tr>
<td>condition</td>
<td>selection status (see the select field of the peer status word)</td>
</tr>
<tr>
<td>last_event</td>
<td>event report (see the event field of the peer status word)</td>
</tr>
<tr>
<td>cnt</td>
<td>event count (see the count field of the peer status word)</td>
</tr>
</tbody>
</table>

clockvar assocID [name [ = value [...]]] [...]
cv assocID [name [ = value [...]]] [...]

Display a list of clock variables for those associations supporting a reference clock.

:config [...]

Send the remainder of the command line, including whitespace, to the server as a run-time configuration command in the same format as the configuration file. This command is experimental until further notice and clarification. Authentication is of course required.

config-from-file <filename>

Send the each line of filename to the server as run-time configuration commands in the same format as the configuration file. This command is experimental until further notice and clarification. Authentication is required.

ifstats

Display statistics for each local network address. Authentication is required.

iostats

Display network and reference clock I/O statistics.

kerninfo

Display kernel loop and PPS statistics. As with other ntpq output, times are in milliseconds. The precision value displayed is in milliseconds as well, unlike the precision system variable.

lassociations

Perform the same function as the associations command, except display mobilized and unmobilized associations.

monstats

Display monitor facility statistics.

mrulist [limited | kod | mincount=count | laddr=localaddr | sort=sortorder | resany=hexmask]

Obtain and print traffic counts collected and maintained by the monitor facility. With the exception of sort=sortorder, the options filter the list returned by ntpd. The limited and kod options return only entries representing client addresses from which the last packet received triggered either discarding or a KoD response. The mincount=count option filters entries representing less than count packets. The laddr=localaddr option filters entries for packets received on any local address other than localaddr. resany=hexmask and resall=hexmask filter entries containing none or less than all, respectively, of the bits in hexmask, which must begin with 0x. The sortorder defaults to lstint and may be any of addr, count, avgint, lstint, or any of those preceded by a minus sign (hyphen) to reverse the sort order. The output columns are:
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lstint</td>
<td>Interval in s between the receipt of the most recent packet from this address and the completion of the retrieval of the MRU list by ntpq.</td>
</tr>
<tr>
<td>avgint</td>
<td>Average interval in s between packets from this address.</td>
</tr>
<tr>
<td>rstr</td>
<td>Restriction flags associated with this address. Most are copied unchanged from the matching restrict command, however 0x400 (kod) and 0x20 (limited) flags are cleared unless the last packet from this address triggered a rate control response.</td>
</tr>
<tr>
<td>r</td>
<td>Rate control indicator, either a period, ( L ) or ( K ) for no rate control response, rate limiting by discarding, or rate limiting with a KoD response, respectively.</td>
</tr>
<tr>
<td>m</td>
<td>Packet mode.</td>
</tr>
<tr>
<td>v</td>
<td>Packet version number.</td>
</tr>
<tr>
<td>count</td>
<td>Packets received from this address.</td>
</tr>
<tr>
<td>rport</td>
<td>Source port of last packet from this address.</td>
</tr>
<tr>
<td>remote address</td>
<td>DNS name, numeric address, or address followed by claimed DNS name which could not be verified in parentheses.</td>
</tr>
</tbody>
</table>

\[
\text{mreadvar } \text{assocID } \langle\text{assocID }angle \text{ variable_name [ = value[ ... ]}
\]

\[
\text{mrv } \text{assocID } \langle\text{assocID }angle \text{ variable_name [ = value[ ... ]}
\]

Perform the same function as the \text{readvar} command, except for a range of association IDs. This range is determined from the association list cached by the most recent \text{associations} command.

\[
\text{passociations}
\]

Perform the same function as the \text{associations} command, except that it uses previously stored data rather than making a new query.

\[
\text{peers} \\
\text{Display a list of peers in the form } [\text{tally}]\text{remote refid st t when pool reach delay offset jitter}
\]

\[
\text{Variable } \text{Description}
\]

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[tally]</td>
<td>single-character code indicating current value of the select field of the peer status word</td>
</tr>
<tr>
<td>remote</td>
<td>host name (or IP number) of peer</td>
</tr>
<tr>
<td>refid</td>
<td>association ID or kiss code</td>
</tr>
<tr>
<td>st</td>
<td>stratum</td>
</tr>
<tr>
<td>t</td>
<td>( u ): unicast or multicast client, ( b ): broadcast or multicast client, ( l ): local (reference clock), ( s ): symmetric (peer), ( A ): manycast server, ( B ): broadcast server, ( M ): multicast server</td>
</tr>
<tr>
<td>when</td>
<td>sec/min/hr since last received packet</td>
</tr>
<tr>
<td>poll</td>
<td>poll interval (log:sub:2 s)</td>
</tr>
<tr>
<td>reach</td>
<td>reach shift register (octal)</td>
</tr>
<tr>
<td>delay</td>
<td>roundtrip delay</td>
</tr>
<tr>
<td>offset</td>
<td>offset of server relative to this host</td>
</tr>
<tr>
<td>jitter</td>
<td>jitter</td>
</tr>
</tbody>
</table>

\[
\text{readvar } \text{assocID name [ = value ] [,...]}
\]

\[
\text{rv } \text{assocID [ name ] [,...]}
\]

Display the specified variables. If \text{assocID} is zero, the variables are from the \text{system variables} name space, otherwise they are from the \text{peer variables} name space. The \text{assocID} is required, as the same name can occur in both spaces. If \text{name} is included, all operative variables in the name space are displayed. In this case only, if the \text{assocID} is omitted, it is assumed zero. Multiple names are specified with comma separators and without whitespace. Note that time values are represented in milliseconds and frequency values in parts-per-million (PPM). Some NTP timestamps are represented in the format YYYYMMDDTTTTT, where YYYY is the year, MM the month of year, DD the day of month and TTTT the time of day.

\[
\text{saveconfig } \langle\text{filename}\rangle
\]

Write the current configuration, including any runtime modifications given with \text{:config} or
config-from-file, to the ntpd host’s file filename. This command will be rejected by the server unless saveconfigdir appears in the ntpd configuration file. filename can use strftime() format specifies to substitute the current date and time, for example, saveconfig ntp-%Y%m%d-%H%M%S.conf. The filename used is stored in system variable savedconfig. Authentication is required.

**writevar** assocID name = value [, ...]
Write the specified variables. If the assocID is zero, the variables are from the system variables name space, otherwise they are from the peer variables name space. The assocID is required, as the same name can occur in both spaces.

**sysinfo**
Display operational summary.

**sysstats**
Print statistics counters maintained in the protocol module.

### 3.2.5 Status Words and Kiss Codes

The current state of the operating program is shown in a set of status words maintained by the system and each association separately. These words are displayed in the rv and as commands both in hexadecimal and decoded short tip strings. The codes, tips and short explanations are on the Event Messages and Status Words page. The page also includes a list of system and peer messages, the code for the latest of which is included in the status word.

Information resulting from protocol machine state transitions is displayed using an informal set of ASCII strings called kiss codes. The original purpose was for kiss-o’-death (KoD) packets sent by the server to advise the client of an unusual condition. They are now displayed, when appropriate, in the reference identifier field in various billboards.

### 3.2.6 System Variables

The following system variables appear in the rv billboard. Not all variables are displayed in some configurations.
The jitter and wander statistics are exponentially-weighted RMS averages. The system jitter is defined in the NTPv4 specification; the clock jitter statistic is computed by the clock discipline module.

When the NTPv4 daemon is compiled with the OpenSSL software library, additional system variables are displayed, including some or all of the following, depending on the particular Autokey dance:

### 3.2.7 Peer Variables

The following peer variables appear in the `rv` billboard for each association. Not all variables are displayed in some configurations.
The bias variable is calculated when the first broadcast packet is received after the calibration volley. It represents the offset of the broadcast subgraph relative to the unicast subgraph. The xleave variable appears only the interleaved symmetric and interleaved modes. It represents the internal queuing, buffering and transmission delays for the preceding packet.

When the NTPv4 daemon is compiled with the OpenSSL software library, additional peer variables are displayed, including the following:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>peer flags (see Autokey specification)</td>
</tr>
<tr>
<td>host</td>
<td>Autokey server name</td>
</tr>
<tr>
<td>flags</td>
<td>peer flags (see Autokey specification)</td>
</tr>
<tr>
<td>signature</td>
<td>OpenSSL digest/signature scheme</td>
</tr>
<tr>
<td>initsequence</td>
<td>initial key ID</td>
</tr>
<tr>
<td>initkey</td>
<td>initial key index</td>
</tr>
<tr>
<td>timestamp</td>
<td>Autokey signature timestamp</td>
</tr>
</tbody>
</table>

### 3.2.8 Clock Variables

The following clock variables appear in the \texttt{cv} billboard for each association with a reference clock. Not all variables are displayed in some configurations.
### 3.3 ntpdc - special NTP query program

#### 3.3.1 Synopsis

```bash
ntpdc [ -46dlnps ] [ -c command ] [ host ] [ ... ]
```

#### 3.3.2 Description

**ntpdc** is deprecated - please use **ntpq** now, as it uses a more sane interface and can provide all of the information that **ntpdc** used to provide.

**ntpdc** is used to query the **ntpd** daemon about its current state and to request changes in that state. The program may be run either in interactive mode or controlled using command line arguments. Extensive state and statistics information is available through the **ntpdc** interface. In addition, nearly all the configuration options which can be specified at startup using **ntpd**’s configuration file may also be specified at run time using **ntpdc**.

If one or more request options are included on the command line when **ntpdc** is executed, each of the requests will be 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, **ntpdc** will attempt to read commands from the standard input and execute these on the NTP server running on the first host given on the command line, again defaulting to localhost when no other host is specified. **ntpdc** will prompt for commands if the standard input is a terminal device.

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

The operation of **ntpdc** are specific to the particular implementation of the **ntpd** daemon and can be expected to work only with this and maybe some previous versions of the daemon. Requests from a remote **ntpdc** program which affect the state of the local server must be authenticated, which requires both the remote program and local server share a common key and key identifier.

Note that in contexts where a host name is expected, a −4 qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a −6 qualifier forces DNS resolution to the IPv6 namespace.
3.3.3 Command Line Options

Specifying a command line option other than -i or -n will cause the specified query (queries) to be sent to the indicated host(s) immediately. Otherwise, ntpdc will attempt to read interactive format commands from the standard input.

-4
   Force DNS resolution of following host names on the command line to the IPv4 namespace.

-6
   Force DNS resolution of following host names on the command line to the IPv6 namespace.

-c <command>
   The following argument is interpreted as an interactive format command and is added to the list of commands to be executed on the specified host(s). Multiple -c options may be given.

-d
   Turn on debugging mode.

-i
   Force ntpdc to operate in interactive mode. Prompts will be written to the standard output and commands read from the standard input.

-l
   Obtain a list of peers which are known to the server(s). This switch is equivalent to -c listpeers.

-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 -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.

3.3.4 Interactive 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, but optionally the output of individual commands may 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 ntpdc program itself and do not result in NTP mode 7 requests being sent to a server. These are described following.

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

delay <milliseconds>
   Specify a time interval to be added to timestamps included in requests which require authentication. This is used to enable (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>
Set the host to which future queries will be sent. Hostname may be either a host name or a numeric address.

hostnames [ yes | no ]
If yes is specified, host names are printed in information displays. If no is specified, numeric addresses are printed instead. The default is yes, unless modified using the command line -n switch.

keyid <keyid>
This command allows the specification of a key number to be used to authenticate configuration requests from ntpdc to the host(s). This must correspond to a key number which the host/server has been configured to use for this purpose (server options: trustedkey, and requestkey). If authentication is not enabled on the host(s) for ntpdc commands, the command "keyid 0" should be given; otherwise the keyid of the next subsequent addpeer/addserver/broadcast command will be used.

quit
Exit ntpdc.

passwd
This command prompts you to type in a password (which will not be echoed) which will be used to authenticate configuration requests. The password must correspond to the key configured for use by the NTP server for this purpose if such requests are to be successful.

timeout <milliseconds>
Specify a timeout period for responses to server queries. The default is about 8000 milliseconds. Note that since ntpdc retries each query once after a timeout, the total waiting time for a timeout will be twice the timeout value set.

3.3.5 Control Message 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
Obtains and prints 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
Obtains 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. The character in the left margin indicates the mode this peer entry is operating in. 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 the server is currently synchronizing to. The contents of the host field may be one of four forms. It may be a host name, an IP address, a reference clock implementation name with its parameter or REFCLK(implementation number, parameter). On hostnames no only IP-addresses will be displayed.

dmpeers
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 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.
**showpeer** peer_address [...]  
Shows 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 [...]  
Show per-peer statistic counters associated with the specified peer(s).

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

**kerninfo**  
Obtain and print kernel phase-lock loop operating parameters. This information is available only if the kernel has been specially modified for a precision timekeeping function.

**loopinfo** [ oneline | multiline ]  
Print the values of selected loop filter variables. The loop filter is the part of NTP which deals with adjusting the local system clock. The offset is the last offset given to the loop filter by the packet processing code. The frequency is the frequency error of the local clock in parts-per-million (ppm). The time const controls the stiffness of the phase-lock loop and thus the speed at which it can adapt to oscillator drift. The watchdog timer value is the number of seconds which have elapsed since the last sample offset was given to the loop filter. The oneline and multiline options specify the format in which this information is to be printed, with multiline as the default.

**sysinfo**  
Print a variety of system state variables, i.e., state related to the local server. All except the last four lines are described in the NTP Version 3 specification, RFC 1305. The system flags show various system flags, some of which can be set and cleared by the enable and disable configuration commands, respectively. These are the auth, bclient, monitor, pll, pps and stats flags. See the ntpd documentation for the meaning of these flags. There are two additional flags which are read only, the kernel_pll and kernel_pps. These flags indicate the synchronization status when the precision time kernel modifications are in use. The kernel_pll indicates that the local clock is being disciplined by the kernel, while the kernel_pps indicates the kernel discipline is provided by the PPS signal. Note that some directives, like enable pps, are only supported on certain versions of ntpd. The stability is the residual frequency error remaining after the system frequency correction is applied and is intended for maintenance and debugging. In most architectures, this value will initially decrease from as high as 500 ppm to a nominal value in the range .01 to 0.1 ppm. If it remains high for some time after starting the daemon, something may be wrong with the local clock, or the value of the kernel variable tick may be incorrect. The broadcastdelay shows the default broadcast delay, as set by the broadcastdelay configuration command. The authdelay shows the default authentication delay, as set by the authdelay configuration command.

**sysstats**  
Print statistics counters maintained in the protocol module.

**memstats**  
Print statistics counters related to memory allocation code.

**iostats**  
Print statistics counters maintained in the input-output module.

**timerstats**  
Print statistics 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 to understand how the restrictions are applied.

**ifstats**  
List interface statistics for interfaces used by ntpd for network communication.
NTP, Release 4.2.8p3

ifreload
Force rescan of current system interfaces. Outputs interface statistics for interfaces that could possibly change.
Marks unchanged interfaces with ., added interfaces with + and deleted interfaces with -.

monlist [ version ]
Obtain and print traffic counts collected and maintained by the monitor facility. The version number should not
normally need to be specified. At most, 600 entries are displayed by monlist. To display the entire MRU list,
use the ntpq program’s mrualist command.

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

3.3.6 Runtime Configuration Requests

All requests which 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 ntpdc. This can be done using the keyid and passwd commands, the latter of which will
prompt at the terminal for a password to use as the encryption key. You will also be prompted automatically for both
the key number and password the first time a command which 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 against 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 will work well with a server on the local host, and may work adequately between time-synchronized hosts on
the same LAN, it will work very 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.

The following commands all make authenticated requests.

addpeer peer_address [ keyid ] [ version ] [ minpoll# | prefer | minpoll N | maxpoll N [ . . . ]]
Add a configured peer association at the given address and operating in symmetric active mode. 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. If the keyid is nonzero, all outgoing packets
to the remote server will have an authentication field attached encrypted with this key. If the value is 0 (or not
given) no authentication will be done. If ntpdc’s key number has not yet been set (e.g., by the keyid command),
it will be set to this value. The version# can be 1 through 4 and defaults to 3. The remaining options are
either a numeric value for minpoll or literals prefer, burst, minpoll N, keyid N, version N, or
maxpoll N (where N is a numeric value), and have the action as specified in the peer configuration file
command of ntpd. See the Server Options page for further information. Each flag (or its absence) replaces
the previous setting. The prefer keyword indicates a preferred peer (and thus will be used primarily for
clock synchronisation if possible). The preferred peer also determines the validity of the PPS signal - if the
preferred peer is suitable for synchronisation so is the PPS signal. The dynamic keyword allows association
configuration even when no suitable network interface is found at configuration time. The dynamic interface
update mechanism may complete the configuration when new interfaces appear (e.g. WLAN/PPP interfaces) at
a later time and thus render the association operable.

addserver peer_address [ address [ keyid ] [ version ] [ minpoll | prefer | iburst | burst
Identical to the addpeer command, except that the operating mode is client.
broadcast peer_address [ keyid ] [ version ] [ prefer ]
Identical to the addpeer command, except that the operating mode is broadcast. In this case a valid non-zero
key identifier and key are required. The peer_address parameter can be the broadcast address of the local
network or a multicast group address assigned to NTP. If a multicast address, a multicast-capable kernel is
required.

unconfig peer_address [...]
This command causes the configured bit to be removed from the specified peer(s). In many cases this will cause
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.

fudge peer_address [ time1 ] [ time2 ] [ stratum ] [ refid ]
This command provides a way to set certain data for a reference clock. See the source listing for further
information.

enable [ auth | bclient | calibrate | kernel | monitor | ntp | pps | stats]
disable [ auth | bclient | calibrate | kernel | monitor | ntp | pps | stats]
These commands operate in the same way as the enable and disable configuration file commands of ntpd.
See the Miscellaneous Options page for further information.

restrict address mask flag [ flag ]
This command operates in the same way as the restrict configuration file commands of ntpd.

unrestrict address mask flag [ flag ]
Unrestrict the matching entry from the restrict list.

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

readkeys
Causes the current set of authentication keys to be purged and a new set to be obtained by rereading the keys
file (which must have been specified in the ntpd configuration file). This allows encryption keys to be changed
without restarting the server.

trustedkey keyid [...] untrustedkey keyid [...] These commands operate in the same way as the trustedkey and untrustedkey configuration file com-
mands of ntpd.

authinfo
Returns information concerning the authentication module, including known keys and counts of encryptions and
decryptions which have been done.

traps
Display the traps set in the server. See the source listing for further information.

addtrap [ address [ port ] [ interface ]]
Set a trap for asynchronous messages. See the source listing for further information.

clrtrap [ address [ port ] [ interface]]
Clear a trap for asynchronous messages. See the source listing for further information.

reset
Clear the statistics counters in various modules of the server. See the source listing for further information.

3.3.7 Bugs

ntpdc is a crude hack. Much of the information it shows is deadly boring and could only be loved by its implementer.
The program was designed so that new (and temporary) features were easy to hack in, at great expense to the program’s
Ease of use. Despite this, the program is occasionally useful.

3.4 ntpdate - set the date and time via NTP

Disclaimer: This program has known bugs and deficiencies and nobody has volunteered to fix them in a long time. The good news is the functionality originally intended for this program is available in the ntpd and sntp programs. See the Deprecating ntpdate topic in the NTP Support wiki for a thorough discussion and analysis of the issues. See the -q command line option in the ntp - Network Time Protocol (NTP) daemon page and/or the sntp - Simple Network Time Protocol (SNTP) Client page. After a suitable period of mourning, the ntpdate program will be retired from this distribution.

3.4.1 Synopsis

```
ntpdate [ -46bBdqsuv ] [ -a key ] [ -e authdelay ] [ -k keyfile ] [ -o version ] [ -p samples ] [ -t timeout ] server [ ... ]
```

3.4.2 Description

`ntpdate` sets the local date and time by polling the Network Time Protocol (NTP) server(s) given as the `server` 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 a subset of the NTP clock filter and selection algorithms are applied to select the best of these. Note that the accuracy and reliability of `ntpdate` depends on the number of servers, the number of polls each time it is run and the interval between runs.

`ntpdate` can be run manually as necessary to set the host clock, or it can be run from the host startup script to set the clock at boot time. This is useful in some cases to set the clock initially before starting the NTP daemon `ntpd`. It is also possible to run `ntpdate` from a `cron` script. However, it is important to note that `ntpdate` with contrived `cron` scripts is no substitute for the NTP daemon, which uses sophisticated algorithms to maximize accuracy and reliability while minimizing resource use. Finally, since `ntpdate` does not discipline the host clock frequency as does `ntpd`, the accuracy using `ntpdate` is limited.

Time adjustments are made by `ntpdate` in one of two ways. If `ntpdate` determines the clock is in error more than 0.5 second it will simply step the time by calling the system `settimeofday()` routine. If the error is less than 0.5 seconds, it will slew the time by calling the system `adjtime()` routine. The latter technique is less disruptive and more accurate when the error is small, and works quite well when `ntpdate` is run by `cron` every hour or two.

`ntpdate` will, if the `-u` flag was not specified, decline to set the date if an NTP server daemon (e.g., `ntpd`) is running on the same host. When running `ntpdate` on a regular basis from `cron` as an alternative to running a daemon, doing so once every hour or two will result in precise enough timekeeping to avoid stepping the clock.

Note that in contexts where a host name is expected, a `-4` qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a `-6` qualifier forces DNS resolution to the IPv6 namespace.

If NetInfo support is compiled into `ntpdate`, then the `server` argument is optional if `ntpdate` can find a time server in the NetInfo configuration for `ntpd`.

3.4.3 Command Line Options

- `-4`
  Force DNS resolution of following host names on the command line to the IPv4 namespace.

- `-6`
  Force DNS resolution of following host names on the command line to the IPv6 namespace.
-a <key>
Enable the authentication function and specify the key identifier to be used for authentication as the argument key. The keys and key identifiers must match in both the client and server key files. The default is to disable the authentication function.

-B
Force the time to always be slewed using the adjtime() system call, even if the measured offset is greater than +500 ms. The default is to step the time using settimeofday() if the offset is greater than ±500 ms. Note that, if the offset is much greater than +500 ms in this case, that it can take a long time (hours) to slew the clock to the correct value. During this time, the host should not be used to synchronize clients.

-b
Force the time to be stepped using the settimeofday() system call, rather than slewed (default) using the adjtime() system call. This option should be used when called from a startup file at boot time.

-d
Enable the debugging mode, in which ntpdate will go through all the steps, but not adjust the local clock and using an unprivileged port. Information useful for general debugging will also be printed.

-e <authdelay>
Specify the processing delay to perform an authentication function as the value authdelay, in seconds and fraction (see ntpd for details). This number is usually small enough to be negligible for most purposes, though specifying a value may improve timekeeping on very slow CPU’s.

-k <keyfile>
Specify the path for the authentication key file as the string keyfile. The default is /etc/ntp.keys. This file should be in the format described in ntpd.

-o <version>
Specify the NTP version for outgoing packets as the integer version, which can be 1, 2, 3 or 4. The default is 4. This allows ntpdate to be used with older NTP versions.

-p <samples>
Specify the number of samples to be acquired from each server as the integer samples, with values from 1 to 8 inclusive. The default is 4.

-q
Query only - don’t set the clock.

-s
Divert logging output from the standard output (default) to the system syslog facility. This is designed primarily for convenience of cron scripts.

-t <timeout>
Specify the maximum time waiting for a server response as the value timeout, in seconds and fraction. The value is is rounded to a multiple of 0.2 seconds. The default is 1 second, a value suitable for polling across a LAN.

-u
Direct ntpdate to use an unprivileged port for outgoing packets. This is most useful when behind a firewall that blocks incoming traffic to privileged ports, and you want to synchronize with hosts beyond the firewall. Note that the -d option always uses unprivileged ports.

-v
Be verbose. This option will cause ntpdate’s version identification string to be logged.

3.4.4 Diagnostics

ntpdate’s exit status is zero if it finds a server and updates the clock, and nonzero otherwise.
3.4.5 Files

/etc/ntp.keys - encryption keys used by ntpdate.

3.4.6 Bugs

The slew adjustment is actually 50% larger than the measured offset, since this (it is argued) will tend to keep a badly drifting clock more accurate. This is probably not a good idea and may cause a troubling hunt for some values of the kernel variables tick and tickadj.

3.5 ntp-wait - waits until ntpd is in synchronized state

3.5.1 Synopsis

ntp-wait [ -v ] [ -n tries ] [ -s seconds ]

3.5.2 Description

The ntp-wait program blocks until ntpd is in synchronized state. This can be useful at boot time, to delay the boot sequence until after “ntpd -g” has set the time.

3.5.3 Command Line Options

-\( n \) <tries>
   Number of tries before giving up. The default is 1000.
-\( s \) <seconds>
   Seconds to sleep between tries. The default is 6 seconds.
-\( v \)
   Be verbose.

3.6 snntp - Simple Network Time Protocol (SNTP) Client

3.6.1 Synopsis

snntp [{--help -?}][{-4 -6}][-a keynum][-b baddress][-B bctpertime][-c][-d][-D debug-level][-g delay][-K kcodefile][-k keyfile][-l logfile][-M steplimit][-o ntpver][-r][-S][-s][-u uctimeout][---wait][---version][address(es)]

3.6.2 Description

This program is a Simple Network Time Protocol (SNTP) client that can be used to query a Network Time Protocol (NTP) server and display the time offset of the system clock relative to the server clock. Run as root it can correct the system clock to this offset as well. It can be run as an interactive command or from a script by a cron job. The program implements the SNTP client protocol defined in RFC 5905, including the full on-wire protocol but does not provide the sanity checks, access controls, security functions and mitigation algorithms as in the full NTP version 4 specification, also defined in RFC 5905.
By default, `sntp` writes the local date and time (i.e., not UTC) to the standard output in the format

```
2011-08-04 00:40:36.642222 (+0000) +0.006611 +/- 0.041061 psp-os1 149.20.68.26
s3 no-leap
```

where the `+0.006611 +/- 0.041061` indicates the time offset and error bound of the system clock relative to the server clock, in seconds. The hostname and/or the IP is displayed, as is the stratum of the server. Finally, the leap indicator status is displayed.

If `-bcaddress` is not specified, the program sends a single message to each address and waits up to `uctimeout` (default 5) seconds for a unicast server response. Otherwise, it sends no message and waits up to `bctimeout` (default 68) seconds for a broadcast NTP message.

### 3.6.3 Options

`sntp` recognizes the following options:

- `-?`, `--help`
  displays usage information. The short form typically requires shell quoting, such as `-\?`, otherwise `?` is consumed by the shell.

- `-4`, `--ipv4`
  When resolving hostnames to IP addresses, use IPv4 addresses only.

- `-6`, `--ipv6`
  When resolving hostnames to IP addresses, use IPv6 addresses only.

- `-a <keynum>, --authentication <keynum>`
  Enable authentication with the key ID `keynum`. `keynum` is a number specified in the keyfile along with an authentication secret (password or digest). See the `-k, --keyfile` option for more details.

- `-b <bcaddress>, --broadcast <bcaddress>`
  Listen for NTP packets sent to the broadcast or multicast address `bcaddress`, which can be a DNS name or IP address. The default maximum time to listen for broadcasts/multicasts, 68 seconds, can be modified with the `-B, --bctimeout` option.

- `-B <bctimeout>, --bctimeout <bctimeout>`
  Wait `bctimeout` seconds for broadcast or multicast NTP message before terminating. The default is 68 seconds, chosen because `ntpd` typically transmits broadcasts/multicasts every 64 seconds. Note that the short option is `-B`, an uppercase letter B.

- `-c`, `--concurrent`
  Concurrently query all addresses returned for hostname. Requests from an NTP client to a single server should never be sent more often than once every two seconds. By default, all addresses resolved from a single hostname are assumed to be for a single instance of `ntpd`, and therefore `sntp` will send queries to these addresses one after another, waiting two seconds between queries. This option indicates multiple addresses returned for a hostname are on different machines, so `sntp` can send concurrent queries. This is appropriate when using `*.pool.ntp.org`, for example.

- `-d`, `--debug-level`
  Increase debug verbosity level by one. May be specified multiple times. See also the `-D, --set-debug-level` option.

- `-D <debug-level>, --set-debug-level <debug-level>`
  Set the debug verbosity level to `debug-level`. The default level is zero. Note that the short option is `-D`, an uppercase letter D. See also the `-d, --debug-level` option.

- `-g <delay>, --gap <delay>`
  Specify the `delay` in milliseconds between outgoing queries, defaulting to 50. `sntp` sends queries to all provided hostnames/addresses in short succession, and by default terminates once the first valid response is received. With
multiple time sources provided, all but one will not be used. To limit the number of queries whose responses will not be used, each query is separated from the preceding one by _delay_ milliseconds, to allow time for responses to earlier queries to be received. A larger _delay_ reduces the query load on the time sources, increasing the time to receive a valid response if the first source attempted is slow or unreachable.

-K <kodfile>, --kod <kodfile>
Specifies the filename _kodfile_ to be used for the persistent history of KoD (Kiss Of Death, or rate-limiting) responses received from servers. The default is _/var/db/ntp-kod_. If the file does not exist, a warning message will be displayed. The file will not be created. Note that the short option is -K, an uppercase letter K.

-k <keyfile>, --keyfile <keyfile>
Specifies the filename _keyfile_ used with the -a/--authentication option. The format of the file is described on the ntp-keygen page.

-l <logfile>, --filelog <logfile>
Specifies the filename in which to append a copy of status messages, which also appear on the terminal.

-M <steplimit>, --steplimit <steplimit>
If both -S/--step and -s/--slew options are provided, an offset of less than _steplimit_ milliseconds will be corrected by slewing the clock using _adjtime_(), while an offset of _steplimit_ or more will be corrected by setting the clock to the corrected time. Note that the short option is -M, an uppercase letter M.

-o <ntpver>, --ntpversion <ntpver>
Specifies the NTP protocol version number _ntpver_ to include in requests, default 4. This option is rarely useful.

-r, --usereservedport
By default, _sntp_ uses a UDP source port number selected by the operating system. When this option is used, the reserved NTP port 123 is used, which most often requires _sntp_ be invoked as the superuser (commonly “root”). This can help identify connectivity failures due to port-based firewalling which affect _ntpd_, which always uses source port 123.

-S, --step
By default, _sntp_ displays the clock offset but does not attempt to correct it. This option enables offset correction by stepping, that is, directly setting the clock to the corrected time. This typically requires _sntp_ be invoked as the superuser (“root”). Note that the short option is -S, an uppercase letter S.

-s, --slew
By default, _sntp_ displays the clock offset but does not attempt to correct it. This option enables offset correction by slewing using _adjtime_(), which changes the rate of the clock for a period long enough to accomplish the required offset (phase) correction. This typically requires _sntp_ be invoked as the superuser (“root”).

-u <uctimeout>, --uctimeout <uctimeout>
Specifies the maximum time _uctimeout_ in seconds to wait for a unicast response before terminating.

--wait
When neither -S/--step nor -s/--slew options are provided, _sntp_ will by default terminate after the first valid response is received. This option causes _sntp_ to instead wait for all pending queries’ responses.

--version
Display the _sntp_ program’s version number and the date and time it was compiled.

### 3.6.4 Return Value

The program returns an exit status of zero for if a valid response is received and non-zero otherwise.
3.6.5 Author

This sntp was originally developed by Johannes Maximilian Kuehn. Harlan Stenn and Dave Hart modified it to query more than one server at a time. See the file ChangeLog in the distribution for details.

3.7 nttrace - trace a chain of NTP servers back to the primary source

3.7.1 Synopsis

nttrace [ -n ] [ -m maxhosts ] [ server ]

3.7.2 Description

nttrace is a perl script that uses the ntpq utility program to follow the chain of NTP servers from a given host back to the primary time source. For nttrace to work properly, each of these servers must implement the NTP Control and Monitoring Protocol specified in RFC 1305 and enable NTP Mode 6 packets.

If given no arguments, nttrace starts with localhost. Here is an example of the output from nttrace:

```
% nttrace
localhost: stratum 4, offset 0.0019529, synch distance 0.144135
server2ozo.com: stratum 2, offset 0.0124263, synch distance 0.115784
usndh.edu: stratum 1, offset 0.0019298, synch distance 0.011993, refid 'WWVB'
```

On each line, the fields are (left to right): the host name, the host stratum, the time offset between that host and the local host (as measured by nttrace; this is why it is not always zero for "localhost"), the host synchronization distance, and (only for stratum-1 servers) the reference clock ID. All times are given in seconds. Note that the stratum is the server hop count to the primary source, while the synchronization distance is the estimated error relative to the primary source. These terms are precisely defined in RFC 1305.

3.7.3 Options

- **-m <max_hosts>**
  
  Sets the upper limit of the number of hosts to check (default: unlimited).

- **-n**
  
  Turns off the printing of host names; instead, host IP addresses are given. This may be useful if a nameserver is down.

3.7.4 Bugs

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

3.8 tickadj - set time-related kernel variables

3.8.1 Synopsis

tickadj [ -Aqs ] [ -a tickadj ] [ -t tick ]
3.8.2 Description

The tickadj program reads, and optionally modifies, several timekeeping-related variables in older kernels that do not have support for precision timekeeping, including HP-UX, SunOS, Ultrix, SGI and probably others. Those machines provide means to patch the kernel /dev/kmem. Newer machines with kernel time support, including Solaris, Tru64, FreeBSD and Linux, should NOT use the program, even if it appears to work, as it will destabilize the kernel time support. Use the npttime program instead.

The particular variables that can be changed with tickadj include tick, which is the number of microseconds added to the system time for a clock interrupt, tickadj, which sets the slew rate and resolution used by the adjtime 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 time-of-day clock or not.

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

Note that tickadj should be run with some caution when being used for the first time on different types of machines. The operations which tickadj tries to perform are not guaranteed to work on all Unix machines and may in rare cases cause the kernel to crash.

3.8.3 Command Line Options

-\texttt{-a \langle tickadj \rangle}
  
  Set the kernel variable \texttt{tickadj} to the value \texttt{``tickadj specified``}.

-\texttt{-A}
  
  Set the kernel variable \texttt{tickadj} to an internally computed “optimal” value.

-\texttt{-t \langle tick \rangle}
  
  Set the kernel variable \texttt{tick} to the value \texttt{``tick``} specified.

-\texttt{-s}
  
  Set the kernel variable \texttt{dosynctodr} to zero, which disables the hardware time-of-year clock, a prerequisite for running the ntpd daemon under SunOS 4.x.

-\texttt{-q}
  
  Normally, tickadj is quite verbose about what it is doing. The \texttt{-q} flag tells it to shut up about everything except errors.

3.8.4 Files

/vmunix /unix /dev/kmem

3.8.5 Bugs

Fiddling with kernel variables at run time as a part of ordinary operations is a hideous practice which is only necessary to make up for deficiencies in the implementation of adjtime in many kernels and/or brokenness of the system clock in some vendors’ kernels. It would be much better if the kernels were fixed and the tickadj program went away.
3.9 ntptime - read and set kernel time variables

3.9.1 Synopsis

ntptime [ -chr ] [ -e est_error ] [ -f frequency ] [ -m max_error ] [ -o offset ] [ -s status ] [ -t time_constant]

3.9.2 Description

This program is useful only with special kernels described in the A Kernel Model for Precision Timekeeping page. It reads and displays time-related kernel variables using the ntp_gettime() system call. A similar display can be obtained using the ntpdc program and kerninfo command.

3.9.3 Options

- `-c`
  Display the execution time of ntptime itself.

- `-e <est_error>`
  Specify estimated error, in microseconds.

- `-f <frequency>`
  Specify frequency offset, in parts per million.

- `-h`
  Display help information.

- `-m <max_error>`
  Specify max possible errors, in microseconds.

- `-o <offset>`
  Specify clock offset, in microseconds.

- `-r`
  Display Unix and NTP times in raw format.

- `-s <status>`
  Specify clock status. Better know what you are doing.

- `-t <time_constant>`
  Specify time constant, an integer in the range 0-10.

3.10 ntp-keygen - generate public and private keys

3.10.1 Synopsis

ntp-keygen [ -deGHIMPT ] [ -b modulus ] [ -c [ RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-SHA1 | RSA-MDC2 | RSA-RIPEMD160 | DSA-SHA | DSA-SHA1 ] ] [ -C cipher ] [-i group ] [ -l days] [ -m modulus ] [ -p passwd1 ] [ -q passwd2 ] [ -S [ RSA | DSA ] ] [ -s host ] [ -V nkeys ]
3.10.2 Description

This program generates cryptographic data files used by the NTPv4 authentication and identity schemes. It can gen-
erate message digest keys used in symmetric key cryptography and, if the OpenSSL software library has been installed,
it can generate host keys, sign keys, certificates, and identity keys and parameters used by the Autokey public key
cryptography. The message digest keys file is generated in a format compatible with NTPv3. All other files are in
PEM-encoded printable ASCII format so they can be embedded as MIME attachments in mail to other sites.

When used to generate message digest keys, the program produces a file containing ten pseudo-random printable
ASCII strings suitable for the MD5 message digest algorithm included in the distribution. If the OpenSSL library is
installed, it produces an additional ten hex-encoded random bit strings suitable for the SHA1 and other message digest
algorithms. The message digest keys file must be distributed and stored using secure means beyond the scope of NTP
itself. Besides the keys used for ordinary NTP associations, additional keys can be defined as passwords for the ntpq
and ntpdc utility programs.

The remaining generated files are compatible with other OpenSSL applications and other Public Key Infrastructure
(PKI) resources. Certificates generated by this program are compatible with extant industry practice, although some
users might find the interpretation of X509v3 extension fields somewhat liberal. However, the identity keys are prob-
ably not compatible with anything other than Autokey.

Some files used by this program are encrypted using a private password. The –p option specifies the password for local
encrypted files and the –q option the password for encrypted files sent to remote sites. If no password is specified, the
host name returned by the Unix gethostname() function, normally the DNS name of the host, is used.

The pw option of the crypto configuration command specifies the read password for previously encrypted local files.
This must match the local password used by this program. If not specified, the host name is used. Thus, if files are
generated by this program without password, they can be read back by ntpd without password, but only on the same
host.

Normally, encrypted files for each host are generated by that host and used only by that host, although exceptions exist
as noted later on this page. The symmetric keys file, normally called ntp.keys, is usually installed in /etc. Other
files and links are usually installed in /usr/local/etc, which is normally in a shared filesystem in NFS-mounted
networks and cannot be changed by shared clients. The location of the keys directory can be changed by the keysdir
configuration command in such cases. Normally, this is in /etc.

This program directs commentary and error messages to the standard error stream stderr and remote files to the
standard output stream stdout where they can be piped to other applications or redirected to files. The names used
for generated files and links all begin with the string ntpkey and include the file type, generating host and filestamp,
as described in the Cryptographic Data Files section below.

3.10.3 Running the Program

To test and gain experience with Autokey concepts, log in as root and change to the keys directory, usually
/usr/local/etc. When run for the first time, or if all files with names beginning ntpkey have been removed,
use the ntp-keygen command without arguments to generate a default RSA host key and matching RSA-MD5
certificate with expiration date one year hence. If run again without options, the program uses the existing keys and
parameters and generates only a new certificate with new expiration date one year hence.

Run the command on as many hosts as necessary. Designate one of them as the trusted host (TH) using ntp-keygen
with the –T option and configure it to synchronize from reliable Internet servers. Then configure the other hosts to
synchronize to the TH directly or indirectly. A certificate trail is created when Autokey asks the immediately ascendant
host towards the TH to sign its certificate, which is then provided to the immediately descendant host on request. All
group hosts should have acyclic certificate trails ending on the TH.

The host key is used to encrypt the cookie when required and so must be RSA type. By default, the host key is also
the sign key used to encrypt signatures. A different sign key can be assigned using the –S option and this can be either
RSA or DSA type. By default, the signature message digest type is MD5, but any combination of sign key type and message digest type supported by the OpenSSL library can be specified using the -c option.

The rules say cryptographic media should be generated with proventic filestamps, which means the host should already be synchronized before this program is run. This of course creates a chicken-and-egg problem when the host is started for the first time. Accordingly, the host time should be set by some other means, such as eyeball-and-wristwatch, at least so that the certificate lifetime is within the current year. After that and when the host is synchronized to a proventic source, the certificate should be re-generated.

Additional information on trusted groups and identity schemes is on the Autokey Public-Key Authentication page.

### 3.10.4 Command Line Options

**-b <modulus>**

Set the modulus for generating identity keys to *modulus* bits. The modulus defaults to 256, but can be set from 256 (32 octets) to 2048 (256 octets). Use the larger moduli with caution, as this can consume considerable computing resources and increases the size of authenticated packets.

**-c [ RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-SHA1 | RSA-MDC2 | RSA-RIPEMD160 | DSA-SHA | DSA-SHA1 ]**

Select certificate digital signature and message digest scheme. Note that RSA schemes must be used with an RSA sign key and DSA schemes must be used with a DSA sign key. The default without this option is RSA-MD5. If compatibility with FIPS 140-2 is required, either the DSA-SHA or DSA-SHA1 scheme must be used.

**-C <cipher>**

Select the OpenSSL cipher to use for password-protected keys. The `openssl -h` command provided with OpenSSL displays available ciphers. The default without this option is des-ede3-cbc.

**-d**

Enable debugging. This option displays the cryptographic data produced for eye-friendly billboards.

**-e**

Extract the IFF or GQ public parameters from the IFFkey or GQkey keys file previously specified. Send the unencrypted data to the standard output stream `stdout`.

**-G**

Generate a new encrypted GQ key file for the Guillou-Quisquater (GQ) identity scheme. This option is mutually exclusive with the -I and -V options.

**-H**

Generate a new encrypted RSA public/private host key file.

**-i <group>**

Set the optional Autokey group name to *group*. This is used in the identity scheme parameter file names. In that role, the default is the host name if no group is provided. The group name, if specified using -i or using -s following an @ character, is also used in certificate subject and issuer names in the form `host@group` and should match the group specified via `crypto ident` or `server ident` in ntpd’s configuration file.

**-I**

Generate a new encrypted IFF key file for the Schnorr (IFF) identity scheme. This option is mutually exclusive with the -G and -V options.

**-l <days>**

Set the lifetime for certificates to *days*. The default lifetime is one year (365 d).

**-m <modulus>**

Set the modulus for generating files to *modulus* bits. The modulus defaults to 512, but can be set from 256 (32 octets) to 2048 (256 octets). Use the larger moduli with caution, as this can consume considerable computing resources and increases the size of authenticated packets.
-M
Generate a new keys file containing 10 MD5 keys and 10 SHA keys. An MD5 key is a string of 20 random printable ASCII characters, while a SHA key is a string of 40 random hex digits. The file can be edited using a text editor to change the key type or key content. This option is mutually exclusive with all other option.

-P
Generate a new private certificate used by the PC identity scheme. By default, the program generates public certificates. Note: the PC identity scheme is not recommended for new installations.

-p <passwd>
Set the password for reading and writing encrypted files to passwd. These include the host, sign and identify key files. By default, the password is the string returned by the Unix gethostname() routine.

-q <passwd>
Set the password for writing encrypted IFF, GQ and MV identity files redirected to stdout to passwd. In effect, these files are decrypted with the -p password, then encrypted with the -q password. By default, the password is the string returned by the Unix gethostname() routine.

-S [ RSA | DSA ]
Generate a new encrypted public/private sign key file of the specified type. By default, the sign key is the host key and has the same type. If compatibility with FIPS 140-2 is required, the sign key type must be DSA.

-T
Generate a trusted certificate. By default, the program generates nontrusted certificates.

-V <nkeys>
Generate nkeys encrypted server keys for the Mu-Varadharajan (MV) identity scheme. This option is mutually exclusive with the -I and -G options. Note: support for this option should be considered a work in progress.

3.10.5 Random Seed File
All cryptographically sound key generation schemes must have means to randomize the entropy seed used to initialize the internal pseudo-random number generator used by the OpenSSL library routines. If a site supports ssh, it is very likely that means to do this are already available. The entropy seed used by the OpenSSL library is contained in a file, usually called .rnd, which must be available when starting the ntp-keygen program or ntpd daemon.

The OpenSSL library looks for the file using the path specified by the RANDFILE environment variable in the user home directory, whether root or some other user. If the RANDFILE environment variable is not present, the library looks for the .rnd file in the user home directory. Since both the ntp-keygen program and ntpd daemon must run as root, the logical place to put this file is in /.rnd or /root/.rnd. If the file is not available or cannot be written, the program exits with a message to the system log.

3.10.6 Cryptographic Data Files
File and link names are in the form ntpkey_key_name.fstamp, where key is the key or parameter type, name is the host or group name and fstamp is the filestamp (NTP seconds) when the file was created). By convention, key names in generated file names include both upper and lower case characters, while key names in generated link names include only lower case characters. The filestamp is not used in generated link names.

The key name is a string defining the cryptographic key type. Key types include public/private keys host and sign, certificate cert and several challenge/response key types. By convention, client files used for challenges have a par subtype, as in the IFF challenge IFFpar, while server files for responses have a key subtype, as in the GQ response GQkey.

All files begin with two nonencrypted lines. The first line contains the file name in the format ntpkey_key_host.fstamp. The second line contains the datestamp in conventional Unix date format. Lines beginning with # are ignored.
The remainder of the file contains cryptographic data encoded first using ASN.1 rules, then encrypted using the DES-CBC algorithm with given password and finally written in PEM-encoded printable ASCII text preceded and followed by MIME content identifier lines.

The format of the symmetric keys file, ordinarily named `ntp.keys`, is somewhat different than the other files in the interest of backward compatibility. Ordinarily, the file is generated by this program, but it can be constructed and edited using an ordinary text editor.

```
# ntpkey_MD5key_deacon.udel.edu.3468889781
# Wed Nov 11 00:56:21 2009
1 MD5 CeR(-`9LRcYr4aO`~?G3A # MD5 key
2 MD 7\v5.o)0Gg[\;??CV1gF # MD5 key
3 MD5 -Sc'>9;@YaErI&6X+7vm # MD5 key
4 MD5 f!.*B^`Yj)??GnB.;DQ # MD5 key
5 MD5 7`I3ng/AohzF9rn+n(i # MD5 key
6 MD5 s'/NO=Q(?f7iQCV+5< # MD5 key
7 MD5 \XE\hq2ra?P|1\F<\F3x # MD5 key
8 MD5 v`S[)jQB`cR!gG;:\%nM # MD5 key
9 MD5 cBVE-KDLw/sFudB}->d # MD5 key
10 MD5 2late4Me # MD5 key
11 SHA1 098f1f8e4abff26e31722c72fcaf7c345833119 # SHA1 key
12 SHA1 da0261d0451785086ff32751713aa225a871f25 # SHA1 key
13 SHA1 61d624c9420c07fb70d1078a065a706c03176ed # SHA1 key
14 SHA1 e15b6f9972701b343c3f147b93c97cb227151 # SHA1 key
15 SHA1 b17b9072e97dcc0221ca86d5d152752669b286 # SEA key
16 MD2 73a92cee3n9576a66a53d3eb267cd31d220d1d67 # MD2 key
17 MD4 4d57a9037544983c44c83a5bebf991ed3d80008e # MD4 key
18 MD5 e8955d03918ac337a7cd826b824c49ce0fa024c0 # MD5 key
19 MDC2 2e4914b67ae4e6c8337ae1ef37d987cde1cf1be # MDC2 key
20 RIFEMD160 a8defc6d2b9a773a3f94553e94e452526ed3514 # RIFEMD160 key
```

Figure 3.1: Typical Symmetric Key File

Figure 3.1 shows a typical symmetric keys file used by the reference implementation. Each line of the file contains three fields, first an integer between 1 and 65534, inclusive, representing the key identifier used in the server and peer configuration commands. Next is the key type for the message digest algorithm, which in the absence of the OpenSSL library must be MD5 to designate the MD5 message digest algorithm. If the OpenSSL library is installed, the key type can be any message digest algorithm supported by that library. However, if compatibility with FIPS 140-2 is required, the key type must be either SHA or SHA1. The key type can be changed using an ASCII text editor. An MD5 key consists of a printable ASCII string less than or equal to 16 characters and terminated by whitespace or a # character. An OpenSSL key consists of a hex-encoded ASCII string of 40 characters, which is truncated as necessary.

Note that the keys used by the `ntpq` and `ntpd` programs are checked against passwords requested by the programs and entered by hand, so it is generally appropriate to specify these keys in human readable ASCII format.

The `ntp-keygen` program generates a MD5 symmetric keys file `ntpkey_MD5key_hostname.filestamp`. Since the file contains private shared keys, it should be visible only to root and distributed by secure means to other subnet hosts. The NTP daemon loads the file `ntp.keys`, so `ntp-keygen` installs a soft link from this name to the generated file. Subsequently, similar soft links must be installed by manual or automated means on the other subnet hosts. While this file is not used with the Autokey Version 2 protocol, it is needed to authenticate some remote configuration commands used by the `ntpq` and `doc:ntpd <ntpd> utilities`.

### 3.10. ntp-keygen - generate public and private keys

39
3.10.7 Bugs

It can take quite a while to generate some cryptographic values, from one to several minutes with modern architectures such as UltraSPARC and up to tens of minutes to an hour with older architectures such as SPARC IPC.

3.11 ntpdsim - Network Time Protocol (NTP) simulator

3.11.1 Synopsis

```
ntpdsim [ -B bdly ] [ -C snse ] [ -O clk_time ] [ -S sim_time ] [ -T ferr ] [ -W fsne ] [ -Y ndly ] [ -X pdly ]
```

3.11.2 Description

The `ntpdsim` program is an adaptation of the `ntpd` operating system daemon. The program operates as a discrete time simulator using specified systematic and random driving sources. It includes all the mitigation and discipline algorithms of the actual daemon, but with the packet I/O and system clock algorithms driven by simulation. Most functions of the real `ntpd` remain intact, including the monitoring, statistics recording, trace and host name resolution features. Further information on the simulator is on the NTP Discrete Event Simulator page.

The simulator is most useful to study NTP behavior in response to time and/or frequency transients under specific conditions of network jitter and oscillator wander. For this purpose the daemon can be driven by pseudorandom jitter and wander sample sequences characteristic of real networks and oscillators. The jitter generator produces samples from a Poisson distribution, while the wander generator produces samples from a Guassian distribution.

The easiest way to use this program is to create a `ntpstats` directory, configuration file `ntp.conf` and frequency file `ntp.drift` and test shell `test.sh` in the base directory. The `ntp.drift` file and `ntpstats` directory can be empty to start. The `test.sh` script can contain something like

```
rm ./ntpstats/*
nptdsim -O 0.1 -C .001 -T 400 -W 1 -c ./ntp.conf,
```

which starts the simulator with a time offset 100 ms, network jitter 1 ms, frequency offset 400 PPM and oscillator wander 1 PPM/s. These parameters represent typical conditions with modern workstations on a Ethernet LAN. The `ntp.conf` file should contain something like

```
disable kernel
server pogo
driftfile ./ntp.drift
statsdir ./ntpstats/
filegen loopstats type day enable
filegen peerstats type day enable
```

3.11.3 Command Line Options

Note: The NTP development team is moving to the use of a syntax-directed configuration file design. When complete these options will be replaced by a new one. Most of the `ntpd` command line options apply also to `ntpdsim`. In addition, the following command line options apply to `ntpdsim`.

- `-B <bdly>`
  Specify beep delay (3600) s.

- `-C <snse>`
  Specify network jitter parameter (0) s.
-O <clk_time>
  Specify initial time offset (0) s.
-S <sim_time>
  Specify simulation duration (86400) s.
-T <ferr>
  Specify initial frequency offset (0) PPM.
-W <fnse>
  Specify oscillator wander parameter (0) PPM/s.
-Y <ndly>
  Specify network propagation delay (.001) s.
-Z <pdly>
  Specify server processing delay (.001) s.

3.11.4 Files
/etc/ntp.conf - the default name of the configuration file
/etc/ntp.drift - the default name of the drift file
/etc/ntp.keys - the default name of the key file

3.12 ntpdsim - Network Time Protocol (NTP) Simulator

3.12.1 Description

The ntpdsim program is used to simulate and study the behavior of an NTP daemon that derives its time from a number of different simulated time sources (servers). Each simulated server can be configured to have a different time offset, frequency offset, propagation delay, processing delay, network jitter and oscillator wander.

The ntpdsim program runs all the same selection, mitigation, and discipline algorithms as the actual ntpd daemon at the client. (It actually uses the same code). However, the input/output routines and servers are simulated. That is, instead of sending the client messages over the network to the actual servers, the client messages are intercepted by the ntpdsim program, which then generates the replies to those messages. The reply messages are carefully “inserted” into the input queue of the client at the right time according to the specified server properties (like propagation delay).

Each simulated server runs according to a specified script that describes the server properties at a particular time. Each script consists of a series of consecutive acts. Each act runs for a particular duration and specifies the frequency offset, propagation delay, network jitter and oscillator wander of the server for that duration. Once the duration of an act expires, the simulated server reconfigures itself according to the properties specified in the next act.

3.12.2 Configuration

The ntpdsim program is configured by providing a configuration file at startup. The crux of the simulator configuration is specified using a simulate command, the syntax of which is given below. Note that all time quantities are in seconds and all frequency quantities are in parts per million (PPM):

<simulate_command> ::= simulate { <init_statement_list> <server_list> }
<init_statement_list> ::= <init_statement_list> <init_statement> | <init_statement>
<init_statement> ::= beep_delay = <number> | simulation_duration = <number>
<server_list> ::= <server_list> <server> | <server>
<server_list> ::= server = <address> { server_offset = <number> ; <act_list> }
<act_list> ::= <act_list> <act> | <act>
<act> ::= <act_stmt_list>
<act_stmt_list> ::= <act_stmt_list> <act_stmt> ; | <act_stmt> ;
<act_stmt> ::= freq_offset = <number> | wander = <number> | jitter = <number> | prop_delay = <number> | proc_delay = <number>

In addition to the simulate command, other standard NTP configuration commands can be specified. These commands have the same meaning as in the ntpd configuration. Note that newlines are not significant within the simulate command even though they are used to mark the end of a normal NTP configuration command. While a newline is an “end of command” terminator for other configuration commands, in the simulate stanza ; (the semicolon) is the “end of command” terminator.

### 3.12.3 Sample Configuration File

A sample ntpdsim configuration file is given below. It specifies two simulated servers, each of which has two acts.

```plaintext
# Client configuration
disable kernel
server pogo
driftfile ./ntp.drift
statsdir ./ntpstats/
filegen loopstats type day enable
filegen peerstats type day enable

# Simulation configuration
simulate {
    simulation_duration = 86400;
    beep_delay = 3600;

    # Server 1
    server = louie.udel.edu {
        server_offset = 0;
        duration = 50000 {
            freq_offset = 400;
            wander = 1.0;
            jitter = 0.001;
            prop_delay = 0.001;
            proc_delay = 0.001;
        };
        duration = 6400 {
            freq_offset = 200;
            wander = 1.0;
            jitter = 0.001;
            prop_delay = 0.001;
            proc_delay = 0.001;
        }
    }

    # Server 2
    server = baldwin.udel.edu {
        server_offset = 0.02;
        duration = 10000 {
```

Chapter 3. Program Manual Pages
freq_offset = 400;
  wander = 1.0;
  jitter = 0.001;
  prop_delay = 0.5;
  proc_delay = 0.001;
}
duration = 60000 {
  freq_offset = 200;
  wander = 1.0;
  jitter = 0.05;
  prop_delay = 0.005;
  proc_delay = 0.001;
}
}
4.1 NTP Version 4 Release Notes

NTP has been under development for almost 30 years, but the paint ain’t dry even now. This release of the NTP Version 4 (NTPv4) distribution for Unix, VMS and Windows incorporates new features and refinements, but retaining backwards compatibility with older versions, including NTPv3 and NTPv2, but not NTPv1. Support for NTPv1 has been discontinued because of certain security vulnerabilities.

4.1.1 New Features

• The behavior of the daemon at startup has been considerably improved. The time to measure the frequency and correct an initial offset error when started for the first time is now no more than ten minutes. Upon restart, it takes no more than five minutes to reduce the initial offset to less than one millisecond without adversely affecting the frequency. This avoids a subsequent frequency correction which could take up to several hours.

• A new feature called interleaved mode can be used in NTP symmetric and broadcast modes. It is designed to improve accuracy by minimizing errors due to queuing and transmission delays. It is described on the NTP Interleaved Modes page.

• The huff-n’-puff filter is designed to avoid large errors with DSL circuits and highly asymmetrical traffic, as when downloading large files. Details are on the The Huff-n’-Puff Filter page.

• A new feature called orphan mode provides an automatic, subnet-wide synchronization feature with multiple sources. It provides reliable backup in isolated networks or in pr when Internet sources have become unavailable. See the Orphan Mode page for further information.

• This release includes comprehensive packet rate management tools to help reduce the level of spurious network traffic and protect the busiest servers from overload. There is support for the optional Kiss-o’-Death (KoD) packet intended to slow down an abusive client. See the Rate Management and the Kiss-o’-Death Packet page for further information.

• There are two new burst mode features available where special conditions apply. One of these is enabled by the iburst keyword in the server configuration command. It is intended for cases where it is important to set the clock quickly when an association is first mobilized. The other is enabled by the burst keyword in the server configuration command. It is intended for cases where the network attachment requires an initial calling or training procedure. See the Association Management page for further information.

• The OpenSSL cryptographic library has replaced the library formerly available from RSA Laboratories. All cryptographic routines except a version of the MD5 message digest algorithm have been removed from the base distribution. All 128-bit and 160-bit message digests algorithms are now supported for both symmetric key and public key cryptosystems. See the Authentication Support page for further information and the Authentication Options page for a list of supported digest algorithms.
• This release includes support for Autokey public-key cryptography for authenticating public servers to clients, as described in RFC 5906. This support requires the –enable-autokey option when building the distribution, which is the default is OpenSSL is available. The deployment of Autokey subnets is now considerably simpler than in earlier versions. A subnet naming scheme is now available to filter manycast and pool configurations. Additional information about Autokey is on the Autokey Public Key Authentication page and links from there.

• The NTP discrete even simulator has been substantially upgraded, now including scenarios with multiple servers and time-sensitive scripts. This allows the NTP algorithms to be tested in an embedded environment with systematic and pseudo-random network delay and oscillator wander distributions. This has been used to verify correct operation under conditions of extreme error and misconfiguration. See the ntpsim - Network Time Protocol (NTP) simulator page. A technical description and performance analysis is given in the white papers at the NTP Project Page.

• NTPv4 includes three new server discovery schemes, which in most applications can avoid per-host configuration altogether. Two of these are based on IP multicast technology, while the remaining one is based on crafted DNS lookups. See the Automatic NTP Configuration Schemes page for further information.

• The status display and event report monitoring functions have been considerably expanded, including new statistics files and event reporting to files and the system log. See the Event Messages and Status Words page for further information.

• Several new options have been added for the ntpd command line. For the inveterate knob twiddlers several of the more important performance variables can be changed to fit actual or perceived special conditions. In particular, the tinker and tos commands can be used to adjust thresholds, throw switches and change limits.

• The ntpd daemon can be operated in a one-time mode similar to ntpdate, which program is headed for retirement. See the ntpd - Network Time Protocol (NTP) daemon page for the new features.

• A number of white papers have been added to the library on the NTP Research Project Page, including:

### 4.1.2 External Links

• Computer Network Time Synchronization - The Network Time Protocol (book)
• NTP Public Services Project (home page)
• NTP Research Project (home page)
• Executive Summary: Computer Network Time Synchronization
• The NTP Timescale and Leap Seconds
• NTP Timestamp Calculations
• The NTP Era and Era Numbering
• Timestamp Capture Principles
• Analysis and Simulation of the NTP On-Wire Protocols
• Time Synchroization for Space Data Links
• NTP Security Analysis
• IEEE 1588 Precision Time Protocol (PTP)
• Autonomous Configuration
• Autonomous Authentication
• Autokey Protocol
• Autokey Identity Schemes
4.1.3 Changes and Upgrades Since the NTPv3 Version (xntp3-5)

This section summarizes general changes since the publication of RFC 1305. Specific changes made during the code upgrade of 2007-2008 are summarized in Historical Notes.

- If the Basic Socket Interface Extensions for IPv6 (RFC 2553) is detected, support for the IPv6 address family is supported in addition to the default support for the IPv4 address family. In contexts where a host name is expected, a -4 qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a -6 qualifier forces DNS resolution to the IPv6 namespace.
- Many changes have been made in the NTP algorithms to improve performance and reliability. A clock state machine has been incorporated to improve behavior under transient conditions. The clock discipline algorithm has been redesigned to improve accuracy, reduce the impact of network disruptions and allow increased poll intervals to 36 hours with only moderate sacrifice in accuracy. The clock select, cluster and combine algorithms have been overhauled as the result of a thorough statistical analysis.
- In all except a very few cases, all timing intervals are randomized, so that the tendency for NTPv3 to self-synchronize and bunch messages, especially with a large number of configured associations, is minimized.
- Support for the precision time kernel modifications, which are now in stock FreeBSD and optional in Linux kernels, is included. With this support the system clock can be disciplined to the order of one nanosecond. The older microtime kernel modifications in Digital/Compaq/HP Tru64, Digital Ultrix and Sun Microsystems SunOS and Solaris, continue to be supported. In either case the support eliminates sawtooth error, which can be in the hundreds of microseconds. Further information is on the Kernel Model for Precision Timekeeping page.
- New reference clock drivers have been added for several GPS receivers now on the market for a total of 44 drivers. The reference clock driver interface is smaller, more rational, more flexible and more accurate. Most of the drivers in NTPv3 have been converted to the NTPv4 interface and continue to operate as before. A summary of the supported drivers is on the Reference Clock Support page. Audio drivers for the Canadian standard time and frequency station CHU, the US standard time and frequency stations WWV/H and for IRIG signals have been updated and capabilities added to allow direct connection of these signals to an audio port. See the Reference Clock Audio Drivers page for further information.
- Support for pulse-per-second (PPS) signals has been extended to all drivers as an intrinsic function. Further information is on the Pulse-Per-Second (PPS) Signal Interfacing page. Typical performance with the PPS interface and a fast machine are in the low microseconds.
- Several small changes have been made to make administration and maintenance more convenience. The entire distribution has been converted to gnu automake, which greatly ease the task of porting to new and different programming environments, as well as reduce the incidence of bugs due to improper handling of idiosyncratic kernel functions. Version control is provided by Bitkeeper using an online repository at www.ntp.org. Trouble ticket reporting is provided using Bugzilla. If ntpd is configured with NetInfo support, it will attempt to read its configuration from the NetInfo service if the default ntp.conf file cannot be read and no file is specified by the -c option. When ntpd starts it looks at the value of umask, and if zero ntpd will set the umask to 022.

4.1.4 Nasty Surprises

There are a few things different about this release that have changed since the latest NTP Version 3 release. Following are a few things to worry about:

- Some configuration commands have been removed, others added and some changed in minor ways. See the Command Index.
- When both IPv4 and IPv6 address families are in use, the host’s resolver library may not choose the intended address family if a server has an IPv4 and IPv6 address associated with the same DNS name. The solution is to use the IPv4 or IPv6 address directly in such cases or use another DNS name that resolves to the intended
address family. Older versions of ntpdc will show only the IPv4 associations with the peers and some other commands. Older versions of ntpq will show 0.0.0.0 for IPv6 associations with the peers and some other commands.

- There is a minor change to the reference ID field of the NTP packet header when operating with IPv6 associations. In IPv4 associations this field contains the 32-bit IPv4 address of the server, in order to detect and avoid loops. In IPv6 associations this field contains the first 32-bits of a MD5 hash formed from the IPv6 address. All programs in the distribution have been modified to work with both address families.

- The tty_clk and ppsclock pulse-per-second (PPS) line discipline/streams modules are no longer supported. The PPS function is now handled by the PPS Clock Discipline driver, which uses the new PPSAPI application program interface adopted by the IETF. Note that the pps configuration file command has been obsoleted by the driver. See the Pulse-Per-Second (PPS) SignalInterfacing page for further information.

- Support for the NTPv1 symmetric mode has been discontinued, since it hasn’t worked for years. Support continues for the NTPv1 client mode, which is used by some SNTP clients.

- The authstuff directory, intended as a development and testing aid for porting cryptographic routines to exotic architectures, has been removed. Testing and conformance validation tools are available in the OpenSSL software distribution.

### 4.2 Command Index

#### 4.2.1 Access Control Commands and Options

- discard - specify headway parameters
- restrict - specify access restrictions

#### 4.2.2 Authentication Commands and Options

- automax - specify Autokey regeneration interval
- controlkey - specify control key ID
- crypto - configure Autokey parameters
- ident - specify Autokey ephemeral group name
- keys - specify symmetric keys filename
- keysdir - specify Autokey key directory
- requestkey - specify request key ID
- revoke - specify Autokey randomization interval
- trustedkey - specify trusted key IDs

#### 4.2.3 Server Commands and Options

- server - configure client association
- peer - configure symmetric peer association
- broadcast - configure broadcast server association
- manycastclient - configure manycast client association
• pool - configure pool association
• unpeer - remove association
• broadcastclient - enable broadcast client
• manycastserver - enable manycast server
• multicastclient - enable multicast client

4.2.4 Monitoring Commands and Options

• filegen - specify monitor files
• statistics - enable writing of statistics records
• statsdir - specify monitor files directory

4.2.5 Reference Clock Commands and Options

• fudge - specify fudge parameters
• server - specify reference clock server

4.2.6 Miscellaneous Commands and Options

• broadcastdelay - specify broadcast delay
• driftfile - specify frequency file
• enable - enable options
• disable - disable options
• includefile - specify include file
• interface - specify which local network addresses to use
• leapfile - specify leapseconds file
• logconfig - configure log file
• mru - control monitor MRU list limits
• phone - specify modem phone numbers
• reset - reset groups of counters
• saveconfigdir - specify saveconfig directory
• setvar - set system variables
• tinker - modify sacred system parameters (dangerous)
• rlimit - alters certain process storage allocation limits
• tos - modify service parameters
• trap - set trap address
• ttl - set time to live
4.3 Access Control Support

The ntpd daemon implements a general purpose access control list (ACL) containing address/match entries sorted first by increasing address values and then by increasing mask values. A match occurs when the bitwise AND of the mask and the packet source address is equal to the bitwise AND of the mask and address in the list. The list is searched in order with the last match found defining the restriction flags associated with the entry.

The ACL is specified as a list of restrict commands in the following format:

```
restrict address [mask mask] [flag][...]
```

The address argument expressed in dotted-quad form is the address of a host or network. Alternatively, the address argument can be a valid host DNS name. The mask argument expressed in IPv4 or IPv6 numeric address form defaults to all mask bits on, 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 for IPv4 and address :: mask :: for IPv6) is always the first entry in the list. restrict default, with no mask option, modifies both IPv4 and IPv6 default entries. restrict source configures a template restriction automatically added at runtime for each association, whether configured, ephemeral, or preemptable, and removed when the association is demobilized.

Some flags have the effect to deny service, some have the effect to enable service and some are conditioned by other flags. The flags are not orthogonal, in that more restrictive flags will often make less restrictive ones redundant. The flags that deny service are classed in two categories, those that restrict time service and those that restrict informational queries and attempts to do run-time reconfiguration of the server.

An example may clarify how it works. Our campus has two class-B networks, 128.4 for the ECE and CIS departments and 128.175 for the rest of campus. Let’s assume (not true!) that subnet 128.4.1 homes critical services like class rosters and spreadsheets. A suitable ACL might look like this:

```
restrict default nopeer # deny new associations
restrict 128.175.0.0 mask 255.255.0.0 # allow campus access
restrict 128.4.0.0 mask 255.255.0.0 none # allow ECE and CIS access
restrict 128.4.1.0 mask 255.255.255.0 notrust # require authentication on subnet 1
restrict time.nist.gov # allow access
```

While this facility may be useful for keeping unwanted, broken or malicious clients from congesting innocent servers, it should not be considered an alternative to the NTP authentication facilities. Source address based restrictions are easily circumvented by a determined cracker.

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).

4.4 Association Management

4.4.1 Association Modes

This page describes the various modes of operation provided in NTPv4. There are three types of associations in NTP: persistent, preemptable and ephemeral. Persistent associations are mobilized by a configuration command and never demobilized. Preemptable associations, which are new to NTPv4, are mobilized by a configuration command which includes the preempt option or upon arrival of an automatic server discovery packet. They are are demobilized by timeout or when preempted by a “better” server, as described on the Automatic Server Discovery Schemes page. Ephemeral associations are mobilized upon arrival of broadcast or multicast server packets and demobilized by timeout.
Ordinarily, successful mobilization of ephemeral associations requires the server to be cryptographically authenticated to the client. This can be done using either symmetric key or Autokey public key cryptography, as described on the Authentication Support page.

There are three principal modes of operation in NTP: client/server, symmetric active/passive and broadcast/multicast. There are three automatic server discovery schemes in NTP: broadcast/multicast, manycast and pool described on the Automatic Server Discovery Schemes page. In addition, the burst options and orphan mode can be used in appropriate cases.

Following is a summary of the operations in each mode. Note that reference to option applies to the commands described on the Server Commands and Options page. See that page for applicability and defaults.

### 4.4.2 Client/Server Mode

Client/server mode is the most common configuration in the Internet today. It operates in the classic remote-procedure-call (RPC) paradigm with stateless servers and stateful clients. In this mode a host sends a client (mode 3) request to the specified server and expects a server (mode 4) reply at some future time. In some contexts this would be described as a “pull” operation, in that the host pulls the time and related values from the server.

A host is configured in client mode using the `server` command and specifying the server DNS name or IPv4 or IPv6 address; the server requires no prior configuration. The `iburst` option described later on this page is recommended for clients, as this speeds up initial synchronization from several minutes to several seconds. The `burst` option described later on this page can be useful to reduce jitter on very noisy dial-up or ISDN network links.

Ordinarily, the program automatically manages the poll interval between the default minimum and maximum values. The `minpoll` and `maxpoll` options can be used to bracket the range. Unless noted otherwise, these options should not be used with reference clock drivers.

### 4.4.3 Symmetric Active/Passive Mode

Symmetric active/passive mode is intended for configurations where a clique of low-stratum peers operate as mutual backups for each other. Each peer operates with one or more primary reference sources, such as a reference clock, or a set of secondary (stratum, 2) servers known to be reliable and authentic. Should one of the peers lose all reference sources or simply cease operation, the other peers will automatically reconfigure so that time and related values can flow from the surviving peers to all hosts in the subnet. In some contexts this would be described as a “push-pull” operation, in that the peer either pulls or pushes the time and related values depending on the particular configuration.

A symmetric active peer sends a symmetric active (mode 1) message to a designated peer. If a matching configured symmetric active association is found, the designated peer returns a symmetric active message. If no matching association is found, the designated peer mobilizes a ephemeral symmetric passive association and returns a symmetric passive (mode 2) message. Since an intruder can impersonate a symmetric active peer and cause a spurious symmetric passive association to be mobilized, symmetric passive mode should always be cryptographically validated.

A peer is configured in symmetric active mode using the `peer` command and specifying the other peer DNS name or IPv4 or IPv6 address. The `burst` and `iburst` options should not be used in symmetric modes, as this can upset the intended symmetry of the protocol and result in spurious duplicate or dropped messages.

As symmetric modes are most often used as root servers for moderate to large subnets where rapid response is required, it is generally best to set the minimum and maximum poll intervals of each root server to the same value using the `minpoll` and `maxpoll` options.

### 4.4.4 Broadcast/Multicast Modes

NTP broadcast and multicast modes are intended for configurations involving one or a few servers and a possibly very large client population. Broadcast mode can be used with Ethernet, FDDI and WiFi spans interconnected by hubs or...
switches. Ordinarily, broadcast packets do not extend beyond a level-3 router. Where service is intended beyond a level-3 router, multicast mode can be used. Additional information is on the Automatic NTP Configuration Options page.

A server is configured to send broadcast or multicast messages using the broadcast command and specifying the subnet address for broadcast or the multicast group address for multicast. A broadcast client is enabled using the broadcastclient command, while a multicast client is enabled using the multicastclient command and specifying the multicast group address. Multiple commands of either type can be used. However, the association is not mobilized until the first broadcast or multicast message is actually received.

### 4.4.5 Manycast and Pool Modes

Manycast and pool modes are automatic discovery and configuration paradigms new to NTPv4. They are intended as a means for a client to troll the nearby network neighborhood to find cooperating willing servers, validate them using cryptographic means and evaluate their time values with respect to other servers that might be lurking in the vicinity. The intended result is that each client mobilizes ephemeral client associations with some number of the “best” of the nearby servers, yet automatically reconfigures to sustain this number of servers should one or another fail. Additional information is on the Automatic Server Discovery Schemes page.

### 4.4.6 Poll Interval Management

NTP uses an intricate heuristic algorithm to automatically control the poll interval for maximum accuracy consistent with minimum network overhead. The algorithm measures the incidental offset and jitter to determine the best poll interval. When ntpd starts, the interval is the default minimum 64 s. Under normal conditions when the clock discipline has stabilized, the interval increases in steps to the default maximum 1024 s. In addition, should a server become unreachable after some time, the interval increases in steps to the maximum in order to reduce network overhead. Additional information about the algorithm is on the Poll Program page.

The default poll interval range is suitable for most conditions, but can be changed using options on the Server Commands and Options and Miscellaneous Options pages. However, when using maximum intervals much larger than the default, the residual clock frequency error must be small enough for the discipline loop to capture and correct. The capture range is 500 PPM with a 64-s interval decreasing by a factor of two for each interval doubling. At a 36-hr interval, for example, the capture range is only 0.24 PPM.

In the NTPv4 specification and reference implementation, the poll interval is expressed in \( \log_2 \) units, properly called the poll exponent. It is constrained by the lower limit minpoll and upper limit maxpoll options of the server command. The limits default to 6 (64 s) and 10 (1024 s), respectively, which are appropriate for the vast majority of cases.

As a rule of thumb, the expected errors increase by a factor of two as the poll interval increases by a factor of four. The poll interval algorithm slowly increases the poll interval when jitter dominates the error budget, but quickly reduces the interval when wander dominates it. More information about this algorithm is on the How NTP Works page.

There is normally no need to change the poll limits, as the poll interval is managed automatically as a function of prevailing jitter and wander. The most common exceptions are the following.

- With fast, lightly loaded LANs and modern processors, the nominal Allan intercept is about 500 s. In these cases the expected errors can be further reduced using a poll exponent of 4 (16 s). In the case of the pulse-per-second (PPS) driver, this is the recommended value.

- With symmetric modes the most stable behavior results when both peers are configured in symmetric active mode with matching poll intervals of 6 (64 s).

- The poll interval should not be modified for reference clocks, with the single exception the ACTS telephone modem driver. In this case the recommended minimum and maximum intervals are 12 (1.1 hr) and 17 (36 hr), respectively.
4.4.7 Burst Options

Occasionally it is necessary to send packets temporarily at intervals less than the poll interval. For instance, with the burst and iburst options of the server command, the poll program sends a burst of several packets at 2-s intervals. In either case the poll program avoids sending needless packets if the server is not responding. The client begins a burst with a single packet. When the first packet is received from the server, the client continues with the remaining packets in the burst. If the first packet is not received within 64 s, it will be sent again for two additional retries before beginning backoff. The result is to minimize network load if the server is not responding. Additional details are on the Poll Program page.

There are two burst options where a single poll event triggers a burst. They should be used only with the server and pool commands, but not with reference clock drivers nor symmetric mode peers. In both modes, received server packets update the clock filter, which selects the best (most accurate) time values. When the last packet in the burst is sent, the next received packet updates the system variables and adjusts the system clock as if only a single packet exchange had occurred.

The iburst option is useful where the system clock must be set quickly or when the network attachment requires an initial calling or training sequence, as in PPP or ISDN services. In general, this option is recommended for server and pool commands. A burst is sent only when the server is unreachable; in particular, when first starting up. Ordinarily, the clock is set within a few seconds after the first received packet. See the Clock State Machine page for further details about the startup behavior.

The burst option is useful in cases of severe network jitter or when the network attachment requires an initial calling or training sequence. This option is recommended when the minimum poll exponent is larger than 10 (1024 s). A burst is sent only when the server is reachable. The number of packets in the burst is determined by the poll interval so that the average interval between packets (headway) is no less than the minimum poll interval for the association.

4.5 Authentication Support

This page describes the various cryptographic authentication provisions in NTPv4. Authentication support allows the NTP client to verify that servers are in fact known and trusted and not intruders intending accidentally or intentionally to masquerade as a legitimate server. A detailed discussion of the NTP multi-layer security model and vulnerability analysis is in the white paper NTP Security Analysis.

The NTPv3 specification (RFC 1305) defined an authentication scheme properly described as symmetric key cryptography. It used the Data Encryption Standard (DES) algorithm operating in cipher-block chaining (CBC) mode. Subsequently, this algorithm was replaced by the RSA Message Digest 5 (MD5) algorithm commonly called keyed-MD5. Either algorithm computes a message digest or one-way hash which can be used to verify the client has the same message digest as the server. The MD5 message digest algorithm is included in the distribution, so without further cryptographic support, the distribution can be freely exported.

If the OpenSSL cryptographic library is installed prior to building the distribution, all message digest algorithms included in the library may be used, including SHA and SHA1. However, if conformance to FIPS 140-2 is required, only a limited subset of these algorithms can be used. This library is available from http://www.openssl.org and can be installed using the procedures outlined in the Building and Installing the Distribution page. Once installed, the configure and build process automatically detects the library and links the library routines required.

In addition to the symmetric key algorithms, this distribution includes support for the Autokey public key algorithms and protocol specified in RFC 5906 “Network Time Protocol Version 4: Autokey Specification”. This support is available only if the OpenSSL library has been installed and the --enable-autokey option is used when the distribution is built.

Public key cryptography is generally considered more secure than symmetric key cryptography, since the security is based on private and public values which are generated by each participant and where the private value is never revealed. Autokey uses X.509 public certificates, which can be produced by commercial services, the OpenSSL application program, or the ntp-keygen utility program in the NTP software distribution.
Note that according to US law, NTP binaries including OpenSSL library components, including the OpenSSL library itself, cannot be exported outside the US without license from the US Department of Commerce. Builders outside the US are advised to obtain the OpenSSL library directly from OpenSSL, which is outside the US, and build outside the US.

Authentication is configured separately for each association using the `key` or `autokey` option of the `server` configuration command, as described in the Server Options page. The `ntp-keygen` page describes the files required for the various authentication schemes. Further details are in the briefings, papers and reports at the NTP project page linked from www.ntp.org.

By default, the client sends non-authenticated packets and the server responds with non-authenticated packets. If the client sends authenticated packets, the server responds with authenticated packets if correct, or a crypto-NAK packet if not. In the case of unsolicited packets which might consume significant resources, such as broadcast or symmetric mode packets, authentication is required, unless overridden by a `disable auth` command. In the current climate of targeted broadcast or “letterbomb” attacks, defeating this requirement would be decidedly dangerous. In any case, the `notrust` flag, described on the Access Control Options page, can be used to disable access to all but correctly authenticated clients.

### 4.5.1 Symmetric Key Cryptography

The original NTPv3 specification ([RFC 1305](https://tools.ietf.org/html/rfc1305)), as well as the current NTPv4 specification ([RFC 5905](https://tools.ietf.org/html/rfc5905)), allows any one of possibly 65,534 message digest keys (excluding zero), each distinguished by a 32-bit key ID, to authenticate an association. The servers and clients involved must agree on the key ID, key type and key to authenticate NTP packets.

The message digest is a cryptographic hash computed by an algorithm such as MD5 or SHA. When authentication is specified, a message authentication code (MAC) is appended to the NTP packet header. The MAC consists of a 32-bit key identifier (key ID) followed by a 128- or 160-bit message digest. The algorithm computes the digest as the hash of a 128- or 160-bit message digest key concatenated with the NTP packet header fields with the exception of the MAC. On transmit, the message digest is computed and inserted in the MAC. On receive, the message digest is computed and compared with the MAC. The packet is accepted only if the two MACs are identical. If a discrepancy is found by the client, the client ignores the packet, but raises an alarm. If this happens at the server, the server returns a special message called a crypto-NAK. Since the crypto-NAK is protected by the loopback test, an intruder cannot disrupt the protocol by sending a bogus crypto-NAK.

Keys and related information are specified in a keys file, which must be distributed and stored using secure means beyond the scope of the NTP protocol itself. Besides the keys used for ordinary NTP associations, additional keys can be used as passwords for the `ntpq` and `ntpd` utility programs. Ordinarily, the `ntp.keys` file is generated by the `ntp-keygen` program, but it can be constructed and edited using an ordinary text editor.

Each line of the keys file consists of three fields: a key ID in the range 1 to 65,534, inclusive, a key type, and a message digest key consisting of a printable ASCII string less than 40 characters, or a 40-character hex digit string. If the OpenSSL library is installed, the key type can be any message digest algorithm supported by the library. If the OpenSSL library is not installed, the only permitted key type is MD5.

Figure 4.1 shows a typical keys file used by the reference implementation when the OpenSSL library is installed. In this figure, for key IDs in the range 1-10, the key is interpreted as a printable ASCII string. For key IDs in the range 11-20, the key is a 40-character hex digit string. The key is truncated or zero-filled internally to either 128 or 160 bits, depending on the key type. The line can be edited later or new lines can be added to change any field. The key can be change to a password, such as `2late4Me` for key ID 10. Note that two or more keys files can be combined in any order as long as the key IDs are distinct.

When `ntpd` is started, it reads the keys file specified by the `keys` command and installs the keys in the key cache. However, individual keys must be activated with the `trustedkey` configuration command before use. This allows, for instance, the installation of possibly several batches of keys and then activating a key remotely using `ntpq` or `ntpd`. The `requestkey` command selects the key ID used as the password for the `ntpd` utility, while the `controlkey` command selects the key ID used as the password for the `ntpq` utility.
4.5.2 Microsoft Windows Authentication

In addition to the above means, ntpd now supports Microsoft Windows MS-SNTP authentication using Active Directory services. This support was contributed by the Samba Team and is still in development. It is enabled using the mssntp flag of the restrict command described on the Access Control Options page.

Caution: Potential users should be aware that these services involve a TCP connection to another process that could potentially block, denying services to other users. Therefore, this flag should be used only for a dedicated server with no clients other than MS-SNTP.

4.5.3 Public Key Cryptography

See the Autokey Public-Key Authentication page.

4.6 Performance Metrics

This page describes several statistics provided in the NTP specification and reference implementation and how they determine the accuracy and error measured during routine and exceptional operation. These statistics provide the following information.

- Nominal estimate of the server clock time relative to the client clock time. This is called clock offset symbolized by the Greek letter $\theta$.

- Roundtrip system and network delay measured by the on-wire protocol. This is called roundtrip delay symbolized by the Greek letter $\delta$.
• Potential clock offset error due to the maximum uncorrected system clock frequency error. This is called dispersion symbolized by the Greek letter $\epsilon$.

• Expected error, consisting of the root mean square (RMS) nominal clock offset sample differences in a sliding window of several samples. This is called jitter symbolized by the Greek letter $\phi$.

Figure 4.2 shows how the various measured statistics are collected and compiled to calibrate NTP performance.

The data represented in boxes labeled Server are contained in fields in packet received from the server. The data represented in boxes labeled Peer are computed by the on-wire protocol, as described below. The algorithms of the box labeled Selection and Combining Algorithms process the peer data to select a system peer. The System box represents summary data inherited from the system peer. These data are available to application programs and dependent downstream clients.

### 4.6.1 Statistics Summary

Each NTP synchronization source is characterized by the offset $\theta$ and delay $\delta$ samples measured by the on-wire protocol, as described on the How NTP Works page. In addition, the dispersion $\epsilon$ sample is initialized with the sum of the source precision $\rho_R$ and the client precision $\rho$ (not shown) as each source packet is received. The dispersion increases at a rate of 15 $\mu$s/s after that. For this purpose, the precision is equal to the latency to read the system clock. The offset, delay and dispersion are called the sample statistics.

Note. In very fast networks where the client clock frequency is not within 1 PPM or so of the the server clock frequency, the roundtrip delay may have small negative values. This is usually a temporary condition when the client is first started. When using the roundtrip delay in calculations, negative values are assumed zero.

In a window of eight (offset, delay, dispersion) samples, the algorithm described on the Clock Filter Algorithm page selects the sample with minimum delay, which generally represents the most accurate offset statistic. The selected offset sample determines the peer offset and peer delay statistics. The peer dispersion is a weighted average of the dispersion samples in the window. These quantities are recalculated as each update is received from the source. Between updates, both the sample dispersion and peer dispersion continue to grow at the same rate, 15 $\mu$s/s. Finally, the peer jitter $\phi$ is determined as the RMS differences between the offset samples in the window relative to the selected offset sample. The peer statistics are recorded by the `peerstats` option of the `filegen` command. Peer variables are displayed by the `rv` command of the `ntpq` program.

The clock filter algorithm continues to process updates in this way until the source is no longer reachable. Reachability is determined by an eight-bit shift register, which is shifted left by one bit as each poll packet is sent, with 0 replacing the vacated rightmost bit. Each time a valid update is received, the rightmost bit is set to 1. The source is considered reachable if any bit is set to 1 in the register; otherwise, it is considered unreachable. When a source becomes unreachable, a dummy sample with “infinite” dispersion is inserted in the filter window at each poll, thus displacing old samples. This causes the peer dispersion to increase eventually to infinity.
The composition of the source population and the system peer selection is redetermined as each update from each source is received. The system peer and system variables are determined as described on the Mitigation Rules and the prefer Keyword page. The system variables \( \Theta, \Delta, E \) and \( \Phi \) are updated from the system peer variables of the same name and the system stratum set one greater than the system peer stratum. The system statistics are recorded by the loopstats option of the filegen command. System variables are displayed by the \( rv \) command of the ntpq program.

Although it might seem counterintuitive, a cardinal rule in the selection process is, once a sample has been selected by the clock filter algorithm, older samples are no longer selectable. This applies also to the clock select algorithm. Once the peer variables for a source have been selected, older variables of the same or other sources are no longer selectable. The reason for these rules is to limit the time delay in the clock discipline algorithm. This is necessary to preserve the optimum impulse response and thus the risetime and overshoot.

This means that not every sample can be used to update the peer variables, and up to seven samples can be ignored between selected samples. This fact has been carefully considered in the discipline algorithm design with due consideration for feedback loop delay and minimum sampling rate. In engineering terms, even if only one sample in eight survives, the resulting sample rate is twice the Nyquist rate at any time constant and poll interval.

### 4.6.2 Quality of Service

This section discusses how an NTP client determines the system performance using a peer population including reference clocks and remote servers. This is determined for each peer from two statistics, \( \text{peer jitter} \) and \( \text{root distance} \). Peer jitter is determined from various jitter components as described above. It represents the expected error in determining the clock offset estimate. Root distance represents the maximum error of the estimate due to all causes.

The root distance statistic is computed as one-half the root delay of the primary source of time; i.e., the reference clock, plus the root dispersion of that source. The root variables are included in the NTP packet header received from each source. At each update the root delay is recomputed as the sum of the root delay in the packet plus the peer delay, while the root dispersion is recomputed as the sum of the root dispersion in the packet plus the peer dispersion.

Note. In order to avoid timing loops, the root distance is adjusted to the maximum of the above computation and a minimum threshold. The minimum threshold defaults to 1 ms, but can be changed according to client preference using the mindist option of the tos command.

A source is considered selectable only if its root distance is less than the select threshold, by default 1.5 s, but can be changed according to client preference using the maxdist option of the tos command. When an upstream server loses all sources, its root distance apparent to dependent clients continues to increase. The clients are not aware of this condition and continue to accept synchronization as long as the root distance is less than the select threshold.

The root distance statistic is used by the select, cluster and mitigation algorithms. In this respect, it is sometimes called the synchronization distance often shortened simply to distance. The root distance is also used in the following ways.

- Root distance defines the maximum error of the clock offset estimate due to all causes as long as the source remains reachable.
- Root distance defines the upper and lower limits of the correctness interval. This interval represents the maximum clock offset for each of possibly several sources. The clock select algorithm computes the intersection of the correctness intervals to determine the truechimers from the selectable source population.
- Root distance is used by the clock cluster algorithm as a weight factor when pruning outliers from the truechimer population.
- The (normalized) reciprocal of the root distance is used as a weight factor by the combine algorithm when computing the system clock offset and system jitter.
- Root distance is used by the mitigation algorithm to select the system peer from among the cluster algorithm survivors.

The root distance thus functions as a metric in the selection and weighting of the various available sources. The strategy is to select the system peer as the source with the minimum root distance and thus the minimum maximum.
error. The reference implementation uses the Bellman-Ford algorithm described in the literature, where the goal is to minimize the root distance. The algorithm selects the system peer, from which the system root delay and system root dispersion are inherited.

The algorithms described on the Mitigation Rules and the prefer Keyword page deliver several important statistics. The system offset and system jitter are weighted averages computed by the clock combine algorithm. System offset is best interpreted as the maximum-likelihood estimate of the system clock offset, while system jitter, also called estimated error, is best interpreted as the expected error of this estimate. System delay is the root delay inherited from the system peer, while system dispersion is the root dispersion plus contributions due to jitter and the absolute value of the system offset.

The maximum system error, or system distance, is computed as one-half the system delay plus the system dispersion. In order to simplify discussion, certain minor contributions to the maximum error statistic are ignored. If the precision time kernel support is available, both the estimated error and maximum error are reported to user programs via the ntp_adjtime() kernel system call. See the Kernel Model for Precision Timekeeping page for further information.

### 4.7 Rate Management and the Kiss-o’-Death Packet

This page describes the various rate management provisions in NTPv4. Some national time metrology laboratories, including NIST and USNO, use the NTP reference implementation in their very busy public time servers. They operate multiple servers behind load-balancing devices to support aggregate rates up to ten thousand packets per second. The servers need to defend themselves against all manner of broken client implementations that can clog the server and network infrastructure. On the other hand, friendly clients need to avoid configurations that can result in unfriendly behavior.

A review of past client abuse incidence shows the most frequent scenario is a broken client that attempts to send packets at rates of one per second or more. On one occasion due to a defective client design [1], over 750,000 clients demonstrated this abuse. There have been occasions where this abuse has persisted for days at a time. These scenarios are the most damaging, as they can threaten not only the victim server but the network infrastructure as well.

There are several features in the reference implementation designed to defend the servers and network against accidental or intentional flood attack. Other features are used to insure that the client is a good citizen, even if configured in unfriendly ways. The ground rules are:

- Send at the lowest rate consistent with the expected accuracy requirements.
- Maintain strict guard time and minimum average headway time, even if multiple burst options and/or the Autokey protocol are operating.
- When the first packet of a burst is sent to a server, do not send further packets until the first packet has been received from the server.
- Upon receiving a Kiss-o’-Death packet (KoD, see below), immediately reduce the sending rate.

Rate management involves four algorithms to manage resources: (1) poll rate control, (2) burst control, (3) average headway time and (4) guard time. The first two algorithms are described on the Poll Program page; the remaining two are described in following sections.

#### 4.7.1 Minimum Headway Time

The headway is defined for each source as the interval between the last packet sent or received and the next packet for that source. The minimum receive headway is defined as the guard time. In the reference implementation, if the receive headway is less than the guard time, the packet is discarded. The guard time defaults to 2 s, but this can be changed using the minimum option of the discard command. By design, the minimum interval between burst and iburst packets sent by any client is 2 s, which does not violate this constraint. Packets sent by other implementations that violate this constraint will be dropped and a KoD packet returned, if enabled.
4.7.2 Minimum Average Headway Time

There are two features in the reference implementation to manage the minimum average headway time between one packet and the next, and thus the maximum average rate for each source. The transmit throttle limits the rate for transmit packets, while the receive discard limits the rate for receive packets. These features make use of a pair of counters: a client output counter for each association and a server input counter for each distinct client IP address. For each packet received, the input counter increments by a value equal to the minimum average headway (MAH) and then decrements by one each second. For each packet transmitted, the output counter increments by the MAH and then decrements by one each second. The default MAH is 8 s, but this can be changed using the `average` option of the `discard` command.

If the `iburst` or `burst` options are present, the poll algorithm sends a burst of packets instead of a single packet at each poll opportunity. The NTPv4 specification requires that bursts contain no more than eight packets. Starting from an output counter value of zero, the maximum counter value, called the ceiling, can be no more than eight times the MAH. However, if the burst starts with a counter value other than zero, there is a potential to exceed the ceiling. This can result from protocol restarts and/or Autokey protocol operations. In these cases the poll algorithm throttles the output rate by computing an additional headway time so that the next packet sent will not exceed the ceiling. Designs such as this are often called leaky buckets.

The reference implementation uses a special most-recently used (MRU) list of entries, one entry for each distinct client IP address found. Each entry includes the IP address, input counter and process time at the last packet arrival. As each packet arrives, the IP source address is compared to the IP address in each entry in turn. If a match is found the entry is removed and inserted first on the list. If the IP source address does not match any entry, a new entry is created and inserted first, possibly discarding the last entry if the list is full. Observers will note this is the same algorithm used for page replacement in virtual memory systems. However, in the virtual memory algorithm the entry of interest is the last, whereas here the entry of interest is the first.

The input counter for the first entry on the MRU list, representing the current input packet, is decreased by the interval since the entry was last referenced, but not below zero. If the input counter is greater than the ceiling, the packet is discarded. Otherwise, the counter is increased by the MAH and the packet is processed. The result is, if the client maintains an average headway greater than the ceiling and transmits no more than eight packets in a burst, the input counter will not exceed the ceiling. Packets sent by other implementations that violate this constraint will be dropped and a KoD packet returned, if enabled.

The reference implementation has a maximum MRU list size of a few hundred entries. The national time servers operated by NIST and USNO have an aggregate packet rate in the thousands of packets per second from many thousands of customers. Under these conditions, the list overflows after only a few seconds of traffic. However, analysis shows that the vast majority of the abusive traffic is due to a tiny minority of the customers, some of which send at over one packet per second. This means that the few seconds retained on the list is sufficient to identify and discard by far the majority of the abusive traffic.

4.7.3 The Kiss-of-Death Packet

Ordinarily, packets denied service are simply dropped with no further action except incrementing statistics counters. Sometimes a more proactive response is needed to cause the client to slow down. A special packet has been created for this purpose called the kiss-o’-death (KoD) packet. KoD packets have leap indicator 3, stratum 0 and the reference identifier set to a four-octet ASCII code. At present, only one code RATE is sent by the server if the limited and kod flags of the restrict command are present and either the guard time or MAH time are violated.

A client receiving a KoD packet is expected to slow down; however, no explicit mechanism is specified in the protocol to do this. In the reference implementation, the server sets the poll field of the KoD packet to the greater of (a) the server MAH and (b) client packet poll field. In response to the KoD packet, the client sets the peer poll interval to the maximum of (a) the client MAH and (b) the server packet poll field. This automatically increases the headway for following client packets.
In order to make sure the client notices the KoD packet, the server sets the receive and transmit timestamps to the transmit timestamp of the client packet. Thus, even if the client ignores all except the timestamps, it cannot do any useful time computations. KoD packets themselves are rate limited to no more than one packet per guard time, in order to defend against flood attacks.

4.7.4 References


4.8 Reference Clock Support

NTP Version 4 supports almost four dozen satellite, radio and telephone modem reference clocks plus several audio devices for instrumentation signals. A general description of the reference clock support is on this page. Additional information about each reference clock driver can be found via links from this page. Additional information is on the Debugging Hints for Reference Clock Drivers and How To Write a Reference Clock Driver pages. Information on how to support pulse-per-second (PPS) signals produced by some devices is on the Pulse-per-second (PPS) Signal Interfacing page. All reference clock drivers require that the reference clock use only Coordinated Universal Time (UTC). Timezone and standard/daylight adjustments are performed by the operating system kernel.

A reference clock will generally (though not always) be a radio timecode receiver synchronized to standard time as provided by NIST and USNO in the US, NRC in Canada and their counterparts elsewhere in the world. A device driver specific to each reference clock must be compiled in the distribution; however, most common radio, satellite and telephone modem clocks are included by default and are activated by configuration commands.

Reference clocks are supported in the same way as ordinary NTP clients and use the same filter, select, cluster and combine algorithms. Drivers have addresses in the form 127.127.*.* where * is the driver type and * is a unit number in the range 0-3 to distinguish multiple instances of the same driver. The connection to the computer is device dependent, usually a serial port, parallel port or special bus peripheral, but some can work directly from an audio codec or sound card. The particular device is specified by adding a soft link from the name used by the driver to the particular device name.

The server command is used to configure a reference clock. Only the mode, minpoll, maxpoll, and prefer options are supported for reference clocks, as described on the Reference Clock Commands page. The prefer option is discussed on the Mitigation Rules and the prefer Keyword page. Some of these options have meaning only for selected clock drivers.

The fudge command can be used to provide additional information for individual drivers and normally follows immediately after the server command. The reference clock stratum is by default 0, so that the server stratum appears to clients as 1. The stratum option can be used to set the stratum to any value in the range 0 through 15. The refid option can be used to change the reference identifier, as might in the case when the driver is disciplined by a pulse-per-second (PPS) source. The device-dependent mode, time and flag options can provide additional driver customization.

4.8.1 Special Considerations

The Audio Drivers page describes three software drivers that process audio signals from an audio codec or sound card. One is for the NIST time and frequency stations WWV and WWVH, another for the Canadian time and frequency station CHU. These require an external shortwave radio and antenna. A third is for the generic IRIG signal produced by some timing devices. Currently, these are supported in FreeBSD, Solaris and SunOS and likely in other system as well.
The **Undisciplined Local Clock** driver can simulate a reference clock when no external synchronization sources are available. If a server with this driver is connected directly or indirectly to the public Internet, there is some danger that it can destabilize other clients. It is not recommended that the local clock driver be used in this way, as the orphan mode described on the **Association Management** page provides a generic backup capability.

The local clock driver can also be used when an external synchronization source such as the IEEE 1588 Precision Time Protocol or NIST Lockclock directly synchronizes the computer time. Further information is on the **External Clock Discipline and the Local Clock Driver** page.

Several drivers make use of the pulse-per-second (PPS) signal discipline, which is part of the generic driver interface, so require no specific configuration. For those drivers that do not use this interface, the **PPS Clock Discipline** driver can be used to provide this function. It normally works in conjunction with the reference clock that produces the timecode signal, but can work with another driver or remote server. When PPS kernel features are present, the driver can redirect the PPS signal to the kernel.

Some drivers depending on longwave or shortwave radio services need to know the radio propagation time from the transmitter to the receiver. This must be calculated for each specific receiver location and requires the geographic coordinates of both the transmitter and receiver. The transmitter coordinates for various radio services are given in the **Time and Frequency Standard Station Information** page. Receiver coordinates can be obtained locally or from Google Earth. The actual calculations are beyond the scope of this document.

Depending on interface type, port speed, etc., a reference clock can have a small residual offset relative to another. To reduce the effects of jitter when switching from one driver to the another, it is useful to calibrate the drivers to a common ensemble offset. The **enable calibrate** configuration command described on the **Miscellaneous Options** page activates a special feature which automatically calculates a correction factor for each driver relative to an association designated the prefer peer.

### 4.8.2 List of Reference Clock Drivers

Following is a list showing the type and title of each driver currently implemented. The compile-time identifier for each is shown in parentheses. Click on a selected type for specific description and configuration documentation, including the clock address, reference ID, driver ID, device name and serial line speed. For those drivers without specific documentation, please contact the author listed in the **Copyright Notice** page.

- **Type 1** Undisciplined Local Clock (`LOCAL`)
- **Type 2** Deprecated: was Trak 8820 GPS Receiver
- **Type 3** PSTI/Traconex 1020 WWV/WWVH Receiver (`WWV_PST`)
- **Type 4** Spectracom WWVB/GPS Receivers (`WWVB_SPEC`)
- **Type 5** TrueTime GPS/GOES/OMEGA Receivers (`TRUETIME`)
- **Type 6** IRIG Audio Decoder (`IRIG_AUDIO`)
- **Type 7** Radio CHU Audio Demodulator/Decoder (`CHU`)
- **Type 8** Generic Reference Driver (`PARSE`)
- **Type 9** Magnavox MX4200 GPS Receiver (`GPS_MX4200`)
- **Type 10** Austron 2200A/2201A GPS Receivers (`GPS_AS2201`)
- **Type 11** Arbiter 1088A/B GPS Receiver (`GPS_ARBITER`)
- **Type 12** KSI/Odetics TPRO/S IRIG Interface (`IRIG_TPRO`)
- **Type 13** Leitch CSD 5300 Master Clock Controller (`ATOM_LEITCH`)
- **Type 14** EES M201 MSF Receiver (`MSF_EES`)
- **Type 15** reserved
• Type 16 Bancomm GPS/IRIG Receiver (GPS_BANCOMM)
• Type 17 Datum Precision Time System (GPS_DATUM)
• Type 18 NIST/USNO/PTB Modem Time Services (ACTS_MODEM)
• Type 19 Heath WWV/WWVH Receiver (WWV_HEATH)
• Type 20 Generic NMEA GPS Receiver (NMEA)
• Type 21 TrueTime GPS-VME Interface (GPS_VME)
• Type 22 PPS Clock Discipline (PPS)
• Type 23 reserved
• Type 24 reserved
• Type 25 reserved
• Type 26 Hewlett Packard 58503A GPS Receiver (GPS_HP)
• Type 27 Arcron MSF Receiver (MSF_ARCRON)
• Type 28 Shared Memory Driver (SHM)
• Type 29 Trimble Navigation Palisade GPS (GPS_PALISADE)
• Type 30 Motorola UT Oncore GPS (GPS_ONCORE)
• Type 31 Rockwell Jupiter GPS (GPS_JUPITER)
• Type 32 Chrono-log K-series WWVB receiver (CHRONOLOG)
• Type 33 Dumb Clock (DUMBCLOCK)
• Type 34 Ultralink WWVB Receivers (ULINK)
• Type 35 Conrad Parallel Port Radio Clock (PCF)
• Type 36 Radio WWV/H Audio Demodulator/Decoder (WWV)
• Type 37 Forum Graphic GPS Dating station (FG)
• Type 38 hopf GPS/DCF77 6021/komp for Serial Line (HOPF_S)
• Type 39 hopf GPS/DCF77 6039 for PCI-Bus (HOPF_P)
• Type 40 JJY Receivers (JJY)
• Type 41 TrueTime 560 IRIG-B Decoder
• Type 42 Zyfer GPStarplus Receiver
• Type 43 RIPE NCC interface for Trimble Palisade
• Type 44 NeoClock4X - DCF77 / TDF serial line
• Type 45 Spectracom TSYNC PCI
• Type 46 GPSD NG client protocol
CHAPTER FIVE

CLIENT AND SERVER CONFIGURATION

5.1 Automatic Server Discovery Schemes

This page describes the automatic server discovery schemes provided in NTPv4. There are three automatic server discovery schemes: broadcast/multicast, many cast, and server pool, which are described on this page. The broadcast/multicast and many cast schemes utilize the ubiquitous broadcast or one-to-many paradigm native to IPv4 and IPv6. The server pool scheme uses DNS to resolve addresses of multiple volunteer servers scattered throughout the world.

All three schemes work in much the same way and might be described as *grab-n'-prune*. Through one means or another they grab a number of associations either directly or indirectly from the configuration file, order them from best to worst according to the NTP mitigation algorithms, and prune the surplus associations.

5.1.1 Association Management

All schemes use an iterated process to discover new preemptable client associations as long as the total number of client associations is less than the *maxclock* option of the *tos* command. The *maxclock* default is 10, but it should be changed in typical configuration to some lower number, usually two greater than the *minclock* option of the same command.

All schemes use a stratum filter to select just those servers with stratum considered useful. This can avoid large numbers of clients ganging up on a small number of low-stratum servers and avoid servers below or above specified stratum levels. By default, servers of all strata are acceptable; however, the *tos* command can be used to restrict the acceptable range from the *floor* option, inclusive, to the *ceiling* option, exclusive. Potential servers operating at the same stratum as the client will be avoided, unless the *cohort* option is present. Additional filters can be supplied using the methods described on the Authentication Support page.

The pruning process uses a set of unreachable counters, one for each association created by the configuration or discovery processes. At each poll interval, the counter is increased by one. If an acceptable packet arrives for a persistent (configured) or ephemeral (broadcast/multicast) association, the counter is set to zero. If an acceptable packet arrives for a preemptable (manycast, pool) association and survives the selection and clustering algorithms, the counter is set to zero. If the counter reaches an arbitrary threshold of 10, the association becomes a candidate for pruning.

The pruning algorithm is very simple. If an ephemeral or preemptable association becomes a candidate for pruning, it is immediately demobilized. If a persistent association becomes a candidate for pruning, it is not demobilized, but its poll interval is set at the maximum. The pruning algorithm design avoids needless discovery/prune cycles for associations that wander in and out of the survivor list, but otherwise have similar characteristics.

Following is a summary of each scheme. Note that reference to option applies to the commands described on the Configuration Options page. See that page for applicability and defaults.
5.1.2 Broadcast/Multicast Scheme

A broadcast server generates messages continuously at intervals by default 64 s and time-to-live by default 127. These
defaults can be overridden by the minpoll and ttl options, respectively. Not all kernels support the ttl option.
A broadcast client responds to the first message received by waiting a randomized interval to avoid implosion at the
server. It then polls the server in client/server mode using the iburst option in order to quickly authenticate the
server, calibrate the propagation delay and set the client clock. This normally results in a volley of six client/server
exchanges at 2-s intervals during which both the synchronization and cryptographic protocols run concurrently.

Following the volley, the server continues in listen-only mode and sends no further messages. If for some reason the
broadcast server does not respond to these messages, the client will cease transmission and continue in listen-only
mode with a default propagation delay. The volley can be avoided by using the broadcastdelay command with
nonzero argument.

A server is configured in broadcast mode using the broadcast command and specifying the broadcast address
of a local interface. If two or more local interfaces are installed with different broadcast addresses, a broadcast
command is needed for each address. This provides a way to limit exposure in a firewall, for example. A broadcast
client is configured using the broadcastclient command.

NTP multicast mode can be used to extend the scope using IPv4 multicast or IPv6 broadcast with defined span.
The IANA has assigned IPv4 multicast address 224.0.1.1 and IPv6 address FF05::101 (site local) to NTP, but these
addresses should be used only where the multicast span can be reliably constrained to protect neighbor networks. In
general, administratively scoped IPv4 group addresses should be used, as described in RFC 2365, or GLOP group
addresses, as described in RFC 2770.

A multicast server is configured using the broadcast command, but specifying a multicast address instead of a
broadcast address. A multicast client is configured using the multicastclient command specifying a list of one
or more multicast addresses. Note that there is a subtle distinction between the IPv4 and IPv6 address families. The
IPv4 broadcast or multicast mode is determined by the IPv4 class. For IPv6 the same distinction can be made using
the link-local prefix FF02 for each interface and site-local prefix FF05 for all interfaces.

It is possible and frequently useful to configure a host as both broadcast client and broadcast server. A number of hosts
configured this way and sharing a common broadcast address will automatically organize themselves in an optimum
configuration based on stratum and synchronization distance.

Since an intruder can impersonate a broadcast server and inject false time values, broadcast mode should always be
cryptographically authenticated. By default, a broadcast association will not be mobilized unless cryptographically
authenticated. If necessary, the auth option of the disable command will disable this feature. The feature can be
selectively enabled using the notrust option of the restrict command.

With symmetric key cryptography each broadcast server can use the same or different keys. In one scenario on a
broadcast LAN, a set of broadcast clients and servers share the same key along with another set that share a different
key. Only the clients with matching key will respond to a server broadcast. Further information is on the Authentication
Support page.

Public key cryptography can be used with some restrictions. If multiple servers belonging to different secure groups
share the same broadcast LAN, the clients on that LAN must have the client keys for all of them. This scenario is
illustrated in the example on the Autokey Public Key Authentication page.

5.1.3 Manycast Scheme

Manycast is an automatic server discovery and configuration paradigm new to NTPv4. It is intended as a means for a
client to troll the nearby network neighborhood to find cooperating servers, validate them using cryptographic means
and evaluate their time values with respect to other servers that might be lurking in the vicinity. It uses the grab-n'-
drop paradigm with the additional feature that active means are used to grab additional servers should the number of
associations fall below the maxclock option of the tos command.
The manycast paradigm is not the anycast paradigm described in RFC 1546, which is designed to find a single server from a clique of servers providing the same service. The manycast paradigm is designed to find a plurality of redundant servers satisfying defined optimality criteria.

A manycast client is configured using the `manycastclient` configuration command, which is similar to the `server` configuration command. It sends ordinary client mode messages, but with a broadcast address rather than a unicast address and sends only if less than `maxclock` associations remain and then only at the minimum feasible rate and minimum feasible time-to-live (TTL) hops. The polling strategy is designed to reduce as much as possible the volume of broadcast messages and the effects of implosion due to near-simultaneous arrival of manycast server messages. There can be as many manycast client associations as different addresses, each one serving as a template for future unicast client/server associations.

A manycast server is configured using the `manycastserver` command, which listens on the specified broadcast address for manycast client messages. If a manycast server is in scope of the current TTL and is itself synchronized to a valid source and operating at a stratum level equal to or lower than the manycast client, it replies with an ordinary unicast server message.

The manycast client receiving this message mobilizes a preemptable client association according to the matching manycast client template. This requires the server to be cryptographically authenticated and the server stratum to be less than or equal to the client stratum.

It is possible and frequently useful to configure a host as both manycast client and manycast server. A number of hosts configured this way and sharing a common multicast group address will automatically organize themselves in an optimum configuration based on stratum and synchronization distance.

The use of cryptographic authentication is always a good idea in any server discovery scheme. Both symmetric key and public key cryptography can be used in the same scenarios as described above for the broadcast-multicast scheme.

### 5.1.4 Server Pool Scheme

The idea of targeting servers on a random basis to distribute and balance the load is not a new one; however, the NTP pool scheme puts this on steroids. At present, several thousand operators around the globe have volunteered their servers for public access. In general, NTP is a lightweight service and servers used for other purposes don’t mind an additional small load. The trick is to randomize over the population and minimize the load on any one server while retaining the advantages of multiple servers using the NTP mitigation algorithms.

To support this service, custom DNS software is used by pool.ntp.org and its subdomains to discover a random selection of participating servers in response to a DNS query. The client receiving this list mobilizes some or all of them, similar to the manycast discovery scheme, and prunes the excess. Unlike `manycastclient`, cryptographic authentication is not required. The pool scheme solicits a single server at a time, compared to `manycastclient` which solicits all servers within a multicast TTL range simultaneously. Otherwise, the pool server discovery scheme operates as manycast does.

The pool scheme is configured using one or more `pool` commands with DNS names indicating the pool from which to draw. The `pool` command can be used more than once; duplicate servers are detected and discarded. In principle, it is possible to use a configuration file containing a single line `pool pool.ntp.org`. The NTP Pool Project offers instructions on using the pool with the `server` command, which is suboptimal but works with older versions of ntpd predating the `pool` command. With recent ntpd, consider replacing the multiple `server` commands in their example with a single `pool` command.
5.2 Server Commands and Options

5.2.1 Server and Peer Addresses

Following is a description of the server configuration commands in NTPv4. There are two classes of commands, configuration commands that configure an association with a remote server, peer or reference clock, and auxiliary commands that specify environment variables that control various related operations.

The various modes described on the Association Management page are determined by the command keyword and the DNS name or IP address. Addresses are classed by type as (s) a remote server or peer (IPv4 class A, B and C or IPv6), (b) the IPv4 broadcast address of a local interface, (m) a multicast address (IPv4 class D or IPv6), or (r) a reference clock address (127.127.x.x). For type m addresses the IANA has assigned the multicast group address IPv4 224.0.1.1 and IPv6 ff05::101 (site local) exclusively to NTP, but other nonconflicting addresses can be used.

If the Basic Socket Interface Extensions for IPv6 (RFC 2553) is detected, support for the IPv6 address family is generated in addition to the default IPv4 address family. IPv6 addresses can be identified by the presence of colons "::" in the address field. IPv6 addresses can be used almost everywhere where IPv4 addresses can be used, with the exception of reference clock addresses, which are always IPv4. Note that in contexts where a host name is expected, a -4 qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a -6 qualifier forces DNS resolution to the IPv6 namespace.

5.2.2 Server Commands

Unless noted otherwise, further information about these commands is on the Association Management page.

These commands specify the remote server name or address to be used and the mode in which to operate. The address can be either a DNS name or a IPv4 or IPv6 address in standard notation. In general, multiple commands of each type can be used for different server and peer addresses or multicast groups.

server address [options ...]
peer address [options ...]
broadcast address [options ...]
manycastclient address [options ...]
pool address [options ...]
unpeer [address | associd]

server
For type s and r addresses (only), this command mobilizes a persistent client mode association with the specified remote server or local reference clock. If the preempt flag is specified, a preemptable client mode association is mobilized instead.

peer
For type s addresses (only), this command mobilizes a persistent symmetric-active mode association with the specified remote peer.

broadcast
For type b and m addresses (only), this command mobilizes a broadcast or multicast server mode association. Note that type b messages go only to the interface specified, but type m messages go to all interfaces.

manycastclient
For type m addresses (only), this command mobilizes a preemptable manycast client mode association for the multicast group address specified. In this mode the address must match the address specified on the manycastserver command of one or more designated manycast servers. Additional information about this command is on the Automatic Server Discovery page.

pool
For type s addresses (only) this command mobilizes a preemptable pool client mode association for the DNS
name specified. The DNS name must resolve to one or more IPv4 or IPv6 addresses. Additional information about this command is on the Automatic Server Discovery page. The www.pool.ntp.org page describes a compatible pool of public NTP servers.

**unpeer**
This command removes a previously configured association. An address or association ID can be used to identify the association. Either an IP address or DNS name can be used. This command is most useful when supplied via ntpq runtime configuration commands: `config` and `config-from-file`.

### 5.2.3 Server Command Options

**autokey**
Send and receive packets authenticated by the Autokey scheme described on the Autokey Public Key Authentication page. This option is mutually exclusive with the `key` option.

**burst**
When the server is reachable, send a burst of packets instead of the usual one. This option is valid only with the `server` command and type `s` addresses. It is a recommended option when the `maxpoll` option is greater than 10 (1024 s). Additional information about this option is on the Poll Program page.

**iburst**
When the server is unreachable, send a burst of packets instead of the usual one. This option is valid only with the `server` command and type `s` addresses. It is a recommended option with this command. Additional information about this option is on the Poll Program page.

**ident <group>**
Specify the group name for the association. See the Autokey Public-Key Authentication page for further information.

**key <key>**
Send and receive packets authenticated by the symmetric key scheme described in the Authentication Support page. The `<key>` specifies the key identifier with values from 1 to 65534, inclusive. This option is mutually exclusive with the `autokey` option.

**minpoll <minpoll>**

**maxpoll <maxpoll>**
These options specify the minimum and maximum poll intervals for NTP messages, in seconds as a power of two. The maximum poll interval defaults to 10 (1024 s), but can be increased by the `maxpoll` option to an upper limit of 17 (36 hr). The minimum poll interval defaults to 6 (64 s), but can be decreased by the `minpoll` option to a lower limit of 3 (8 s). Additional information about this option is on the Poll Program page.

**mode <option>**
Pass the `option` to a reference clock driver, where `option` is an integer in the range from 0 to 255, inclusive. This option is valid only with type `r` addresses.

**noselect**
Marks the server or peer to be ignored by the selection algorithm as unreachable, but visible to the monitoring program. This option is valid only with the `server` and `peer` commands.

**preempt**
Specifies the association as preemptable rather than the default persistent. This option is ignored with the `broadcast` command and is most useful with the `manycastclient` and `pool` commands.

**prefer**
Mark the server as preferred. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. See the Mitigation Rules and the `prefer` Keyword page for further information. This option is valid only with the `server` and `peer` commands.
true
Mark the association to assume truechimer status; that is, always survive the selection and clustering algorithms. This option can be used with any association, but is most useful for reference clocks with large jitter on the serial port and precision pulse-per-second (PPS) signals. Caution: this option defeats the algorithms designed to cast out falsetickers and can allow these sources to set the system clock. This option is valid only with the server and peer commands.

ttl \(<\texttt{ttl}>\)
This option specifies the time-to-live \(\texttt{ttl}\) for the broadcast command and the maximum \(\texttt{ttl}\) for the expanding ring search used by the manycastclient command. Selection of the proper value, which defaults to 127, is something of a black art and should be coordinated with the network administrator. This option is invalid with type \(r\) addresses.

version \(<\texttt{version}>\)
Specifies the version number to be used for outgoing NTP packets. Versions 1-4 are the choices, with version 4 the default.

xleave
Operate in interleaved mode (symmetric and broadcast modes only). Further information is on the NTP Interleaved Modes page.

### 5.2.4 Auxiliary Commands

**broadcastclient**
Enable reception of broadcast server messages to any local interface (type \(b\) address). Ordinarily, upon receiving a broadcast message for the first time, the broadcast client measures the nominal server propagation delay using a brief client/server exchange, after which it continues in listen-only mode. If a nonzero value is specified in the broadcastdelay command, the value becomes the delay and the volley is not executed. Note: the novolley option has been deprecated for future enhancements. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page. Note that the volley is required with public key authentication in order to run the Autokey protocol.

**manycastserver \(<\texttt{address}>\) [...]
Enable reception of manycast client messages (type \(m\) to the multicasts group address(es) (type \(m\)) specified. At least one address is required. Note that, in order to avoid accidental or malicious disruption, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page.

**multicastclient \(<\texttt{address}>\) [...]
Enable reception of multicast server messages to the multicast group address(es) (type \(m\)) specified. Upon receiving a message for the first time, the multicast client measures the nominal server propagation delay using a brief client/server exchange with the server, then enters the broadcast client mode, in which it synchronizes to succeeding multicast messages. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page.

**mdnstries \(<\texttt{number}>\)
If we are participating in mDNS, after we have synched for the first time we attempt to register with the mDNS system. If that registration attempt fails, we try again at one minute intervals for up to \(\texttt{mdnstries}\) times. After all, ntpd may be starting before mDNS. The default value for \(\texttt{mdnstries}\) is 5.
5.3 Access Control Commands and Options

5.3.1 Commands and Options

Unless noted otherwise, further information about these commands is on the Access Control Support page.

**discard [ average <avg> ] [ minimum <min> ] [ monitor <prob> ]**

Set the parameters of the rate control facility which protects the server from client abuse. If the **limited** flag is present in the ACL, packets that violate these limits are discarded. If, in addition, the **kod** flag is present, a kiss-o’-death packet is returned. See the Rate Management page for further information. The options are:

- **average <avg>**
  - Specify the minimum average interpacket spacing (minimum average headway time) in \( \log_2 \) s with default 3.

- **minimum <min>**
  - Specify the minimum interpacket spacing (guard time) in seconds with default 2.

- **monitor <prob>**
  - Specify the probability of being recorded for packets that overflow the MRU list size limit set by **mru maxmem** or **mru maxdepth**. This is a performance optimization for servers with aggregate arrivals of 1000 packets per second or more.

**restrict default [flag][...]**

**restrict source [flag][...]**

**restrict <address> [mask <mask>] [flag][...]**

The **address** argument expressed in dotted-quad form is the address of a host or network. Alternatively, the **address** argument can be a valid host DNS name. The **mask** argument expressed in IPv4 or IPv6 numeric address form defaults to all mask bits on, 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 for IPv4 and address :: mask :: for IPv6) is always the first entry in the list. **restrict default**, with no mask option, modifies both IPv4 and IPv6 default entries. **restrict source** configures a template restriction automatically added at runtime for each association, whether configured, ephemeral, or preemptible, and removed when the association is demobilized. Some flags have the effect to deny service, some have the effect to enable service and some are conditioned by other flags. The flags are not orthogonal, in that more restrictive flags will often make less restrictive ones redundant. The flags that deny service are classed in two categories, those that restrict time service and those that restrict informational queries and attempts to do run-time reconfiguration of the server. One or more of the following flags may be specified:

- **flake**
  - Discard received NTP packets with probability 0.1; that is, on average drop one packet in ten. This is for testing and amusement. The name comes from Bob Braden’s *flakeway*, which once did a similar thing for early Internet testing.

- **ignore**
  - Deny packets of all kinds, including **ntpq** and **ntpdc** queries.

- **kod**
  - Send a kiss-o’-death (KoD) packet if the **limited** flag is present and a packet violates the rate limits established by the **discard** command. KoD packets are themselves rate limited for each source address separately. If the **kod** flag is used in a restriction which does not have the **limited** flag, no KoD responses will result.

- **limited**
  - Deny time service if the packet violates the rate limits established by the **discard** command. This does not apply to **ntpq** and **ntpdc** queries.
**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 requestors being denied service. This flag modifies the assignment algorithm by allowing low priority traps to be overridden by later requests for normal priority traps.

**mssntp**
Enable Microsoft Windows MS-SNTP authentication using Active Directory services. **Note:** Potential users should be aware that these services involve a TCP connection to another process that could potentially block, denying services to other users. Therefore, this flag should be used only for a dedicated server with no clients other than MS-SNTP.

**nomodify**
Deny ntpq and ntpdc queries which attempt to modify the state of the server (i.e., run time reconfiguration). Queries which return information are permitted.

**noquery**
Deny ntpq and ntpdc queries. Time service is not affected.

**nopeer**
Deny packets that might mobilize an association unless authenticated. This includes broadcast, symmetric-active and manycast server packets when a configured association does not exist. It also includes pool associations, so if you want to use servers from a pool directive and also want to use nopeer by default, you’ll want a "restrict source ..." line as well that does not include the nopeer directive. Note that this flag does not apply to packets that do not attempt to mobilize an association.

**noserve**
Deny all packets except ntpq and ntpdc queries.

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

**notrust**
Deny packets that are not cryptographically authenticated. Note carefully how this flag interacts with the auth option of the enable and disable commands. If auth is enabled, which is the default, authentication is required for all packets that might mobilize an association. If auth is disabled, but the notrust flag is not present, an association can be mobilized whether or not authenticated. If auth is disabled, but the notrust flag is present, authentication is required only for the specified address/mask range.

**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). A restrict line containing ntpport is considered more specific than one with the same address and mask, but lacking ntpport.

**version**
Deny packets that do not match the current NTP version.

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).
5.4 Authentication Commands and Options

5.4.1 Commands and Options

Unless noted otherwise, further information about these commands is on the Authentication Support page.

**automax** [<logsec>]

Specifies the interval between regenerations of the session key list used with the Autokey protocol, as a power of 2 in seconds. Note that the size of the key list for each association depends on this interval and the current poll interval. The default interval is 12 (about 1.1 hr). For poll intervals above the specified interval, a session key list with a single entry will be regenerated for every message sent. See the Autokey Public Key Authentication page for further information.

**controlkey** <keyid>

Specifies the key ID for the ntpq utility, which uses the standard protocol defined in RFC 1305. The *keyid* argument is the key ID for a trusted key, where the value can be in the range 1 to 65534, inclusive.

**crypto** [digest <digest>] [host <name>] [ident <name>] [pw <password>] [randfile <file>]

This command activates the Autokey public key cryptography and loads the required host keys and certificate. If one or more files are unspecified, the default names are used. Unless the complete path and name of the file are specified, the location of a file is relative to the keys directory specified in the *keysdir* configuration command with default /usr/local/etc. See the Autokey Public Key Authentication page for further information. Following are the options.

**digest** <digest>

Specify the message digest algorithm, with default MD5. If the OpenSSL library is installed, *digest* can be any message digest algorithm supported by the library. The current selections are: MD2, MD4, MD5, MDC2, RIPEMD160, SHA and SHA1. All participants in an Autokey subnet must use the same algorithm. The Autokey message digest algorithm is separate and distinct from the symmetric key message digest algorithm. Note: If compliance with FIPS 140-2 is required, the algorithm must be ether SHA or SHA1.

**host** <name>

Specify the cryptographic media names for the host, sign and certificate files. If this option is not specified, the default name is the string returned by the Unix *gethostname()* routine.

Note: In the latest Autokey version, this option has no effect other than to change the cryptographic media file names.

**ident** <name>

Specify the cryptographic media names for the identity scheme files. If this option is not specified, the default name is the string returned by the Unix *gethostname()* routine.

Note: In the latest Autokey version, this option has no effect other than to change the cryptographic media file names.

**pw** <password>

Specifies the password to decrypt files previously encrypted by the *ntp-keygen* program with the *-p* option. If this option is not specified, the default password is the string returned by the Unix *gethostname()* routine.

**randfile** <file>

Specifies the location of the random seed file used by the OpenSSL library. The defaults are described on the *ntp-keygen* page.

**ident** <group>

Specifies the group name for ephemeral associations mobilized by broadcast and symmetric passive modes. See
the Autokey Public-Key Authentication page for further information.

**keys <path>**
Specifies the complete directory path for the key file containing the key IDs, key types and keys used by ntpd, ntpq and ntpdc when operating with symmetric key cryptography. The format of the keyfile is described on the ntp-keygen page. This is the same operation as the –k command line option. Note that the directory path for Autokey cryptographic media is specified by the keysdir command.

**keysdir <path>**
Specifies the complete directory path for the Autokey cryptographic keys, parameters and certificates. The default is /usr/local/etc/. Note that the path for the symmetric keys file is specified by the keys command.

**requestkey <keyid>**
Specifies the key ID for the ntpdc utility program, which uses a proprietary protocol specific to this implementation of ntpd. The keyid argument is a key ID for a trusted key, in the range 1 to 65534, inclusive.

**revoke [<logsec>]**
Specifies the interval between re-randomization of certain cryptographic values used by the Autokey scheme, as a power of 2 in seconds, with default 17 (36 hr). See the Autokey Public-Key Authentication page for further information.

**trustedkey [keyid] | (<lowid> ... <highid>) [...]**
Specifies the key ID(s) which are trusted for the purposes of authenticating peers with symmetric key cryptography. Key IDs used to authenticate ntpq and ntpdc operations must be listed here and additionally be enabled with controlkey and/or requestkey. The authentication procedure for time transfer requires that both the local and remote NTP servers employ the same key ID and secret for this purpose, although different keys IDs may be used with different servers. Ranges of trusted key IDs may be specified: trustedkey (1 ... 19) 1000 (100 ... 199) enables the lowest 120 key IDs which start with the digit 1. The spaces surrounding the ellipsis are required when specifying a range.

### 5.5 Reference Clock Commands and Options

#### 5.5.1 Reference Clock Addresses

Unless noted otherwise, further information about these commands is on the Reference Clock Support page.

Reference clocks are identified by a syntactically correct but invalid IP address, in order to distinguish them from ordinary NTP peers. These addresses are of the form 127.127.t.u, where t is an integer denoting the clock type and u indicates the unit number in the range 0-3. While it may seem overkill, it is in fact sometimes useful to configure multiple reference clocks of the same type, in which case the unit numbers must be unique.

#### 5.5.2 Commands and Options

**server 127.127.t.u [prefer] [mode <int>] [minpoll <int>] [maxpoll <int>]**
This command can be used to configure reference clocks in special ways. The options are interpreted as follows:

- **prefer**
  Marks the reference clock as preferred. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. See the Mitigation Rules and the prefer Keyword page for further information.

- **mode <int>**
  Specifies a mode number which is interpreted in a device-specific fashion. For instance, it selects a dialing protocol in the ACTS driver and a device subtype in the parse drivers.

- **minpoll <int>**
maxpoll <int>
These options specify the minimum and maximum polling interval for reference clock messages in log2
seconds. For most directly connected reference clocks, both minpoll and maxpoll default to 6 (64 s).
For modem reference clocks, minpoll is ordinarily set to 10 (about 17 m) and maxpoll to 15 (about 9
h). The allowable range is 4 (16 s) to 17 (36 h) inclusive.

fudge 127.127.t.u [time1 <sec>] [time2 <sec>] [stratum <int>] [refid <string>] [flag1 0|1] [flag2 0|1] [flag3 0|1] [flag4 0|1]
This command can be used to configure reference clocks in special ways. It must immediately follow the
server command which configures the driver. Note that the same capability is possible at run time using the
ntpdc program. The options are interpreted as follows:

time1 <sec>
Specifies a constant to be added to the time offset produced by the driver, a fixed-point decimal number
in seconds. This is used as a calibration constant to adjust the nominal time offset of a particular clock
to agree with an external standard, such as a precision PPS signal. It also provides a way to correct a
systematic error or bias due to serial port or operating system latencies, different cable lengths or receiver
internal delay. The specified offset is in addition to the propagation delay provided by other means, such as
internal DIPswitches. Where a calibration for an individual system and driver is available, an approximate
correction is noted in the driver documentation pages. Note: in order to facilitate calibration when more
than one radio clock or PPS signal is supported, a special calibration feature is available. It takes the form
of an argument to the enable command described in the Miscellaneous Options page and operates as
described in the Reference Clock Support page.

time2 <secs>
Specifies a fixed-point decimal number in seconds, which is interpreted in a driver-dependent way. See the
descriptions of specific drivers in the Reference Clock Support page.

stratum <int>
Specifies the stratum number assigned to the driver in the range 0 to 15, inclusive. This number overrides
the default stratum number ordinarily assigned by the driver itself, usually zero.

refid <string>
Specifies an ASCII string of from one to four characters which defines the reference identifier used by the
driver. This string overrides the default identifier ordinarily assigned by the driver itself.

flag1 flag2 flag3 flag4
These four flags are used for customizing the clock driver. The interpretation of these values, and whether
they are used at all, is a function of the particular driver. However, by convention flag4 is used to enable
recording monitoring data to the clockstats file configured with the filegen command. Additional
information on the filegen command is on the Monitoring Options page.

5.6 Miscellaneous Commands and Options

broadcastdelay <delay>
In broadcast and multicast modes, means are required to determine the network delay between the server and
client. Ordinarily, this is done automatically by the initial calibration exchanges between the client and server.
In some cases, the exchange might not be possible due to network or server access controls. The value of delay is
by default zero, in which case the exchange is enabled. If delay is greater than zero, it becomes the roundtrip
delay (s), as measured by the Unix ping program, and the exchange is disabled.

driftfile <driftfile>
This command specifies the complete path and name of the file used to record the frequency of the local clock
oscillator. This is the same operation as the -f command line option. This command is mutually exclusive
with the freq option of the tinker command. If the file exists, it is read at startup in order to set the initial
frequency and then updated once per hour or more with the current frequency computed by the daemon. If the
file name is specified, but the file itself does not exist, the starts with an initial frequency of zero and creates the
When writing it for the first time. If this command is not given, the daemon will always start with an initial frequency of zero. The file format consists of a single line containing a single floating point number, which records the frequency offset measured in parts-per-million (PPM). The file is updated by first writing the current drift value into a temporary file and then renaming this file to replace the old version.

dscp <dscp>
This command specifies the Differentiated Services Code Point (DSCP) value that is used in sent NTP packets. The default value is 46 for Expedited Forwarding (EF).

enable [auth | bclient | calibrate | kernel | mode7 | monitor | ntp | stats]

disable [auth | bclient | calibrate | kernel | mode7 | monitor | ntp | stats]
Provides a way to enable or disable various system options. Flags not mentioned are unaffected. Note that most of these flags can be modified remotely using ntpq utility program’s :config and config-from-file commands.

auth
Enables the server to synchronize with unconfigured peers only if the peer has been correctly authenticated using either public key or private key cryptography. The default for this flag is enable.

bclient
Enables the server to listen for a message from a broadcast or multicast server, as in the multicastclient command with default address. The default for this flag is disable.

calibrate
Enables the calibrate feature for reference clocks. The default for this flag is disable.

kernel
Enables the kernel time discipline, if available. The default for this flag is enable if support is available, otherwise disable.

mode7
Enables processing of NTP mode 7 implementation-specific requests which are used by the deprecated ntpdc program. The default for this flag is disable. This flag is excluded from runtime configuration using ntpq. The ntpq program provides the same capabilities as ntpdc using standard mode 6 requests.

monitor
Enables the monitoring facility. See the ntpq program and the monstats and mru list commands, as well as the Access Control Options for details. The monitoring facility is also enabled by the presence of limited in any restrict commands. The default for this flag is enable.

ntp
Enables time and frequency discipline. In effect, this switch opens and closes the feedback loop, which is useful for testing. The default for this flag is enable.

stats
Enables the statistics facility. See the Monitoring Options page for further information. The default for this flag is enabled. This flag is excluded from runtime configuration using ntpq.

includefile <includefile>
This command allows additional configuration commands to be included from a separate file. Include files may be nested to a depth of five; upon reaching the end of any include file, command processing resumes in the previous configuration file. This option is useful for sites that run ntpd on multiple hosts, with (mostly) common options (e.g., a restriction list).

interface [listen | ignore | drop] [all | ipv4 | ipv6 | wildcard | name | address[/prefixlen]]
This command controls which network addresses ntpd opens, and whether input is dropped without processing. The first parameter determines the action for addresses which match the second parameter. That parameter specifies a class of addresses, or a specific interface name, or an address. In the address case, prefixlen determines how many bits must match for this rule to apply. ignore prevents opening matching addresses, drop causes
ntpd to open the address and drop all received packets without examination. Multiple interface commands can be used. The last rule which matches a particular address determines the action for it. interface commands are disabled if any -I, --interface, -L, or --novirtualips command-line options are used. If none of those options are used and no interface actions are specified in the configuration file, all available network addresses are opened. The nic command is an alias for interface.

leapfile <leapfile>
This command loads the NIST leapseconds file and initializes the leapsecond values for the next leapsecond time, expiration time and TAI offset. The file can be obtained directly from NIST national time servers using ftp as the ASCII file pub/leap-seconds. The leapfile is scanned when ntpd processes the leapfile directive or when ntpd detects that leapfile has changed. ntpd checks once a day to see if the leapfile has changed. While not strictly a security function, the Autokey protocol provides means to securely retrieve the current or updated leapsecond values from a server.

leapsmearinterval <seconds>
This EXPERIMENTAL option is only available if ntpd was built with the --enable-leap-smear option to the configure script. It specifies the interval over which a leap second correction will be applied. Recommended values for this option are between 7200 (2 hours) and 86400 (24 hours). DO NOT USE THIS OPTION ON PUBLIC-ACCESS SERVERS! See https://bugs.ntp.org/2855 for more information.

logconfig <configkeyword>
This command controls the amount and type of output written to the system syslog facility or the alternate logfile log file. All configkeyword keywords can be prefixed with =, + and −, where = sets the syslogmask, + adds and − removes messages. syslog messages can be controlled in four classes (clock, peer, sys and sync). Within these classes four types of messages can be controlled: informational messages (info), event messages (events), statistics messages (statistics) and status messages (status). Configuration keywords are formed by concatenating the message class with the event class. The all prefix can be used instead of a message class. A message class may also be followed by the all keyword to enable/disable all messages of the respective message class. By default, logconfig output is set to allsync. Thus, a minimal log configuration could look like this: logconfig=syncstatus +sysevents. This would just list the synchronizations state of ntpd and the major system events. For a simple reference server, the following minimum message configuration could be useful: logconfig=syncall +clockall This configuration will list all clock information and synchronization information. All other events and messages about peers, system events and so on is suppressed.

logfile <logfile>
This command specifies the location of an alternate log file to be used instead of the default system syslog facility. This is the same operation as the -l command line option.

mru [maxdepth <count> | maxmem <kilobytes> | mindepth <count> | maxage <seconds> | initialloc <count> | initialmem <kilobytes> | incalloc <count> | incmem <kilobytes>]
Controls size limits of the monitoring facility Most Recently Used (MRU) list of client addresses, which is also used by the rate control facility.

maxdepth <count>
maxmem <kilobytes>
Equivalent upper limits on the size of the MRU list, in terms of entries or kilobytes. The actual limit will be up to incalloc entries or incmem kilobytes larger. As with all of the mru options offered in units of entries or kilobytes, if both maxdepth and maxmem are used, the last one used controls. The default is 1024 kilobytes.

mindepth <count>
Lower limit on the MRU list size. When the MRU list has fewer than mindepth entries, existing entries are never removed to make room for newer ones, regardless of their age. The default is 600 entries.

maxage <seconds>
Once the MRU list has mindepth entries and an additional client address is to be added to the list, if the oldest entry was updated more than maxage seconds ago, that entry is removed and its storage reused.
the oldest entry was updated more recently, the MRU list is grown, subject to maxdepth/maxmem. The default is 64 seconds.

**initalloc** <count>

**initmem** <kilobytes>
Initial memory allocation at the time the monitoring facility is first enabled, in terms of entries or kilobytes. The default is 4 kilobytes.

**incalloc** <count>

**incmem** <kilobytes>
Size of additional memory allocations when growing the MRU list, in entries or kilobytes. The default is 4 kilobytes.

**nonvolatile** <threshold>
Specify the threshold in seconds to write the frequency file, with default of 1e-7 (0.1 PPM). The frequency file is inspected each hour. If the difference between the current frequency and the last value written exceeds the threshold, the file is written and the threshold becomes the new threshold value. If the threshold is not exceeded, it is reduced by half. This is intended to reduce the frequency of unnecessary file writes for embedded systems with nonvolatile memory.

**phone** <dial> ...
This command is used in conjunction with the ACTS modem driver (type 18). The arguments consist of a maximum of 10 telephone numbers used to dial USNO, NIST or European time services. The Hayes command ATDT is normally prepended to the number, which can contain other modem control codes as well.

**reset** [allpeers] [auth] [ctl] [io] [mem] [sys] [timer]
Reset one or more groups of counters maintained by ntpd and exposed by ntpq and ntpdc.

**rlimit** [memlock <Nmegabytes> | stacksize <N4kPages> | filenum <Nfiledescriptors>]
This command alters certain process storage allocation limits, and is only available on some operating systems. Options are as follows:

**memlock** <Nmegabytes>
Specify the number of megabytes of memory that can be allocated. Probably only available under Linux, this option is useful when dropping root (the -i option). The default is 32 megabytes. Setting this to zero will prevent any attempt to lock memory.

**stacksize** <N4kPages>
Specifies the maximum size of the process stack on systems with the mlockall() function. Defaults to 50 4k pages (200 4k pages in OpenBSD).

**filenum** <Nfiledescriptors>
Specifies the maximum number of file descriptors ntp may have open at the same time. Defaults to system default.

**saveconfigdir** <directory_path>
Specify the directory in which to write configuration snapshots requested with ntpq’s saveconfig command. If saveconfigdir does not appear in the configuration file, saveconfig requests are rejected by ntpd.

**setvar** <variable> [default]
This command adds an additional system variable. These variables can be used to distribute additional information such as the access policy. If the variable of the form name = value is followed by the default keyword, the variable will be listed as part of the default system variables (ntpq rv command). These additional variables serve informational purposes only. They are not related to the protocol other that they can be listed. The known protocol variables will always override any variables defined via the setvar mechanism. There are three special variables that contain the names of all variable of the same group. The sys_var_list holds the names of all system variables. The peer_var_list holds the names of all peer variables and the clock_var_list holds the names of the reference clock variables.
This command alters certain system variables used by the clock discipline algorithm. The default values of these variables have been carefully optimized for a wide range of network speeds and reliability expectations. Very rarely is it necessary to change the default values; but, some folks can’t resist twisting the knobs. Options are as follows:

- **allan** `<allan>`
  Specifies the Allan intercept, which is a parameter of the PLL/FLL clock discipline algorithm, in seconds with default 1500 s.

- **dispersion** `<dispersion>`
  Specifies the dispersion increase rate in parts-per-million (PPM) with default 15 PPM.

- **freq** `<freq>`
  Specifies the frequency offset in parts-per-million (PPM). This option is mutually exclusive with the drift-file command.

- **huffpuff** `<huffpuff>`
  Specifies the huff-n’-puff filter span, which determines the most recent interval the algorithm will search for a minimum delay. The lower limit is 900 s (15 min), but a more reasonable value is 7200 (2 hours). See the Huff-n’-Puff Filter page for further information.

- **panic** `<panic>`
  Specifies the panic threshold in seconds with default 1000 s. If set to zero, the panic sanity check is disabled and a clock offset of any value will be accepted.

- **step** `<step>`
  Specifies the step threshold in seconds. The default without this command is 0.128 s. If set to zero, step adjustments will never occur. Note: The kernel time discipline is disabled if the step threshold is set to zero or greater than 0.5 s. Further details are on the Clock State Machine page.

- **stepout** `<stepout>`
  Specifies the stepout threshold in seconds. The default without this command is 300 s. Since this option also affects the training and startup intervals, it should not be set less than the default. Further details are on the Clock State Machine page.

This command alters certain system variables used by the clock selection and clustering algorithms. The default values of these variables have been carefully optimized for a wide range of network speeds and reliability expectations. Very rarely is it necessary to change the default values; but, some folks can’t resist twisting the knobs. It can be used to select the quality and quantity of peers used to synchronize the system clock and is most useful in dynamic server discovery schemes. The options are as follows:

- **beacon** `<beacon>`
  The manycast server sends packets at intervals of 64 s if less than maxclock servers are available. Otherwise, it sends packets at the beacon interval in seconds. The default is 3600 s. See the Automatic Server Discovery page for further details.

- **ceiling** `<ceiling>`
  Specify the maximum stratum (exclusive) for acceptable server packets. The default is 16. See the Automatic Server Discovery page for further details.

- **cohort** `{ 0 | 1 }`
  Specify whether (1) or whether not (0) a server packet will be accepted for the same stratum as the client. The default is 0. See the Automatic Server Discovery page for further details.

- **floor** `<floor>`
  Specify the minimum stratum (inclusive) for acceptable server packets. The default is 1. See the Automatic Server Discovery page for further details.
maxclock <maxclock>
  Specify the maximum number of servers retained by the server discovery schemes. The default is 10. See the Automatic Server Discovery page for further details.

maxdist <maxdistance>
  Specify the synchronization distance threshold used by the clock selection algorithm. The default is 1.5 s. This determines both the minimum number of packets to set the system clock and the maximum roundtrip delay. It can be decreased to improve reliability or increased to synchronize clocks on the Moon or planets.

minclock <minclock>
  Specify the number of servers used by the clustering algorithm as the minimum to include on the candidate list. The default is 3. This is also the number of servers to be averaged by the combining algorithm.

mindist <mindistance>
  Specify the minimum distance used by the selection and anticlockhop algorithm. Larger values increase the tolerance for outliers; smaller values increase the selectivity. The default is .001 s. In some cases, such as reference clocks with high jitter and a PPS signal, it is useful to increase the value to insure the intersection interval is always nonempty.

minsane <minsane>
  Specify the number of servers used by the selection algorithm as the minimum to set the system clock. The default is 1 for legacy purposes; however, for critical applications the value should be somewhat higher but less than minclock.

orphan <stratum>
  Specify the orphan stratum with default 16. If less than 16 this is the stratum assumed by the root servers. See the Orphan Mode page for further details.

orphanwait <delay>
  Specify the delay in seconds from the time all sources are lost until orphan parent mode is enabled with default 300 s (five minutes). During this period, the local clock driver and the modem driver are not selectable, unless marked with the prefer keyword. This allows time for one or more primary sources to become reachable and selectable before using backup sources, and avoids transient use of the backup sources at startup.

trap host_address [port <port_number>] [interface <interface_address>]
  This command configures a trap receiver at the given host address and port number for 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 of the local interface the message is sent through. Note that on a multihomed host the interface used may vary from time to time with routing changes. The trap receiver will generally log 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 will ensure that no messages are lost when the server is started.

ttl <hop> ...
  This command specifies a list of TTL values in increasing order. up to 8 values can be specified. In manycast mode these values are used in turn in an expanding-ring search. The default is eight multiples of 32 starting at 31.

5.7 Monitoring Commands and Options

5.7.1 Naming Conventions

The ntpd includes a comprehensive monitoring facility which collects statistical data of various types and writes the data to files associated with each type at defined events or intervals. The files associated with a particular type are collectively called the generation file set for that type. The files in the file set are the members of that set.
File sets have names specific to the type and generation epoch. The names are constructed from three concatenated elements prefix, filename and suffix:

**prefix**
The directory path specified in the `statsdir` command.

**name**
The name specified by the `file` option of the `filegen` command.

**suffix**
A string of elements beginning with . (dot) followed by a number of elements depending on the file set type.

Statistics files can be managed using scripts, examples of which are in the `./scripts` directory. Using these or similar scripts and Unix `cron` jobs, the files can be automatically summarized and archived for retrospective analysis.

### 5.7.2 Monitoring Commands and Options

Unless noted otherwise, further information about these commands is on the Event Messages and Status Words page.

```
filegen <name> [file <filename>] [type <type>] [link | nolink] [enable | disable]
```

- **<name>**
  Specifies the file set type from the list in the next section.

- **file <filename>**
  Specifies the filename prefix. The default is the file set type, such as “loopstats”.

- **type <typename>**
  Specifies the file set interval. The following intervals are supported with default day:

  - **none**
    The file set is actually a single plain file.

  - **pid**
    One file set member is created for every incarnation of `ntpd`. The file name suffix is the string .n, where n is the process ID of the `ntpd` server process.

  - **day**
    One file set member is created per day. A day is defined as the period between 00:00 and 23:59 UTC. The file name suffix is the string .yyyyymmd, where yyyy is the year, mm the month of the year and dd the day of the month. Thus, member created on 10 December 1992 would have suffix .19921210.

  - **week**
    One file set member is created per week. The week is defined as the day of year modulo 7. The file name suffix is the string .yyyyWww, where yyyy is the year, W stands for itself and www the week number starting from 0. For example, The member created on 10 January 1992 would have suffix .1992W1.

  - **month**
    One file set member is created per month. The file name suffix is the string .yyyymm, where yyyy is the year and mm the month of the year starting from 1. For example, The member created on 10 January 1992 would have suffix .199201.

  - **year**
    One file set member is generated per year. The file name suffix is the string .yyyy, where yyyy is the year. For example, The member created on 1 January 1992 would have suffix .1992.
One file set member is generated every 24 hours of `ntpd` operation. The filename suffix is the string `.adddddddd`, where `a` stands for itself and `dddddddd` is the `ntpd` running time in seconds at the start of the corresponding 24-hour period.

`link` | `nolink`
--- | ---
It is convenient to be able to access the current file set members by file name, but without the suffix. This feature is enabled by `link` and disabled by `nolink`. If enabled, which is the default, a hard link from the current file set member to a file without suffix is created. When there is already a file with this name and the number of links to this file is one, it is renamed by appending a dot, the letter `C`, and the pid of the `ntpd` 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.

`enable` | `disable`
--- | ---
Enable or disable the recording function, with default `enable`. These options are intended for remote configuration commands.

`statistics <name>...`
Enables writing of statistics records. Currently, eight kinds of statistics are supported: `name(s)` specify the file set type(s) from the list in the next section.

`statsdir <directory_path>`
Specify the directory path prefix for statistics file names.

### 5.7.3 File Set Types

**clockstats**
Record reference clock statistics. Each update received from a reference clock driver appends one line to the `clockstats` file set:

```
49213 525.624 127.127.4.1 93 226 00:08:29.606 D
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49213</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>525.624</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>127.127.4.1</td>
<td>IP</td>
<td>reference clock address</td>
</tr>
<tr>
<td>message</td>
<td>text</td>
<td>log message</td>
</tr>
</tbody>
</table>

The `message` field includes the last timecode received in decoded ASCII format, where meaningful. In some cases a good deal of additional information is displayed. See information specific to each reference clock for further details.

**cryptostats**
Record significant events in the Autokey protocol. This option requires the OpenSSL cryptographic software library. Each event appends one line to the `cryptostats` file set:

```
49213 525.624 128.4.1.1 message
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49213</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>525.624</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>128.4.1.1</td>
<td>IP</td>
<td>source address (0.0.0.0 for system)</td>
</tr>
<tr>
<td>message</td>
<td>text</td>
<td>log message</td>
</tr>
</tbody>
</table>

The `message` field includes the message type and certain ancillary information. See the Authentication Options page for further information.

**loopstats**
Record clock discipline loop statistics. Each system clock update appends one line to the `loopstats` file set:
50935 75440.031 0.000006019 13.778 0.000351733 0.013380 6

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>50935</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>75440.031</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>0.000006019</td>
<td>s</td>
<td>clock offset</td>
</tr>
<tr>
<td>13.778</td>
<td>PPM</td>
<td>frequency offset</td>
</tr>
<tr>
<td>0.000351733</td>
<td>s</td>
<td>RMS jitter</td>
</tr>
<tr>
<td>0.013380</td>
<td>PPM</td>
<td>RMS frequency jitter (aka wander)</td>
</tr>
<tr>
<td>6</td>
<td>log₂s</td>
<td>clock discipline loop time constant</td>
</tr>
</tbody>
</table>

**peerstats**
Record peer statistics. Each NTP packet or reference clock update received appends one line to the `peerstats` file set:

<table>
<thead>
<tr>
<th>Item</th>
<th>MJD</th>
<th>Date</th>
</tr>
</thead>
</table>
| 48773    | 10847.650 | 127.127.4.1 9714 -0.001605376 0.000000000 0.001424877 0.000958674

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>48773</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>10847.650</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>127.127.4.1</td>
<td>IP</td>
<td>source address</td>
</tr>
<tr>
<td>9714</td>
<td>hex</td>
<td>status word</td>
</tr>
<tr>
<td>-0.001605376</td>
<td>s</td>
<td>clock offset</td>
</tr>
<tr>
<td>0.000000000</td>
<td>s</td>
<td>roundtrip delay</td>
</tr>
<tr>
<td>0.001424877</td>
<td>s</td>
<td>dispersion</td>
</tr>
<tr>
<td>0.000958674</td>
<td>s</td>
<td>RMS jitter</td>
</tr>
</tbody>
</table>

The status field is encoded in hex format as described in Appendix B of the NTP specification RFC 1305.

**protostats**
Record significant peer, system and rtpcp; events. Each significant event appends one line to the `protostats` file set:

<table>
<thead>
<tr>
<th>Item</th>
<th>MJD</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>49213</td>
<td>525.624</td>
<td>128.4.1.1 963a 8a message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49213</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>525.624</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>128.4.1.1</td>
<td>IP</td>
<td>source address (0.0.0.0 for system)</td>
</tr>
<tr>
<td>963a</td>
<td>code</td>
<td>status word</td>
</tr>
<tr>
<td>8a</td>
<td>code</td>
<td>event message code</td>
</tr>
<tr>
<td>message</td>
<td>text</td>
<td>event message</td>
</tr>
</tbody>
</table>

The event message code and message field are described on the Event Messages and Status Words page.

**rawstats**
Record timestamp statistics. Each NTP packet received appends one line to the `rawstats` file set:

<table>
<thead>
<tr>
<th>Item</th>
<th>MJD</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>56285</td>
<td>54575.160</td>
<td>128.4.1.1 192.168.1.5 3565350574.400229473 3565350574.442385200 3565350574.44236000 3565350575.154505763 0 4 4 1 8 -21 0.000000 0.000320 .PPS.</td>
</tr>
</tbody>
</table>
### sysstats

Record system statistics. Each hour one line is appended to the `sysstats` file set in the following format:

```
509282132.54336008196509546565125401041471
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>50928</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>2132.543</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>3600</td>
<td>s</td>
<td>time since reset</td>
</tr>
<tr>
<td>81965</td>
<td>#</td>
<td>packets received</td>
</tr>
<tr>
<td>0</td>
<td>#</td>
<td>packets for this host</td>
</tr>
<tr>
<td>9546</td>
<td>#</td>
<td>current versions</td>
</tr>
<tr>
<td>56</td>
<td>#</td>
<td>old version</td>
</tr>
<tr>
<td>512</td>
<td>#</td>
<td>access denied</td>
</tr>
<tr>
<td>540</td>
<td>#</td>
<td>bad length or format</td>
</tr>
<tr>
<td>10</td>
<td>#</td>
<td>bad authentication</td>
</tr>
<tr>
<td>4</td>
<td>#</td>
<td>declined</td>
</tr>
<tr>
<td>147</td>
<td>#</td>
<td>rate exceeded</td>
</tr>
<tr>
<td>1</td>
<td>#</td>
<td>kiss-o’-death packets sent</td>
</tr>
</tbody>
</table>

### timingstats

(Only available when the deamon is compiled with process time debugging support (`–enable-debug-timing -` costs performance). Record processing time statistics for various selected code paths.)

```
5387636.92010.0.3.510.00014592 input processing delay
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>53876</td>
<td>MJD</td>
<td>date</td>
</tr>
<tr>
<td>36.920</td>
<td>s</td>
<td>time past midnight</td>
</tr>
<tr>
<td>10.0.3.5</td>
<td>IP</td>
<td>server address</td>
</tr>
<tr>
<td>1</td>
<td>#</td>
<td>event count</td>
</tr>
<tr>
<td>0.000014592</td>
<td>s</td>
<td>total time</td>
</tr>
<tr>
<td>message</td>
<td>text</td>
<td>code path description (see source)</td>
</tr>
</tbody>
</table>
6.1 External Clock Discipline and the Local Clock Driver

The NTPv4 implementation includes provisions for an external clock, where the system clock is implemented by some external hardware device. One implementation might take the form of a bus peripheral with a high resolution counter disciplined by a GPS receiver, for example. Another implementation might involve another synchronization protocol, such as the Digital Time Synchronization Service (DTSS), where the system time is disciplined to this protocol and NTP clients of the server obtain synchronization indirectly via the server. A third implementation might be a completely separate clock discipline algorithm and synchronization protocol, such as the Lockclock algorithm used with NIST Automated Computer Time Service (ACTS) modem synchronized time.

When external clocks are used in conjunction with NTP service, some way needs to be provided for the external clock driver and NTP daemon ntpd to communicate and determine which discipline is in control. This is necessary in order to provide backup, for instance if the external clock or protocol were to fail and synchronization service fall back to other means, such as a local reference clock or another NTP server. In addition, when the external clock and driver are in control, some means needs to be provided for the clock driver to pass on status information and error statistics to the NTP daemon.

Control and monitoring functions for the external clock and driver are implemented using the Local Clock (type 1) driver and the ntp_adjtime() system call. This system call is implemented by special kernel provisions included in the kernel of several operating systems, including Solaris, Tru64, FreeBSD and Linux, and possibly others. When the external clock is disabled or not implemented, the system call is used to pass time and frequency information, as well as error statistics, to the kernel. Besides disciplining the system time, the same interface can be used by other applications to determine the operating parameters of the discipline.

When the external clock is enabled, ntpd does not discipline the system clock, nor does it maintain the error statistics. In this case, the external clock and driver do this using mechanisms unknown to ntpd; however, in this case the kernel state variables are retrieved at 64-s intervals by the Local Clock driver and used by the clock selection and mitigation algorithms to determine the system variables presented to other NTP clients and peers. In this way, downstream clients and servers in the NTP subnet can make an intelligent choice when more than one server is available.

In order to implement a reliable mitigation between ordinary NTP sources and the external clock source, a protocol is necessary between the local clock driver and the external clock driver. This is implemented using Boolean variables and certain bits in the kernel clock status word. The Boolean variables include the following:

- ntp_enable. set/reset by the enable command. enables ntpd clock discipline
- ntp_control. set during initial configuration if kernel support is available
- kern_enable Set/reset by the enable command

If the kern_enable switch is set, the daemon computes the offset, frequency, maximum error, estimated error, time constant and status bits, then provides them to the kernel via ntp_adjtime(). If this switch is not set, these values are not passed to the kernel; however, the daemon retrieves their present values and uses them in place of the values computed by the daemon.
The `pps_update` bit set in the protocol routine if the prefer peer has survived and has offset less than 128 ms; otherwise set to zero.

The `PPS control` updated to the current time by kernel support if the PPS signal is enabled and working correctly. Set to zero in the adjust routine if the interval since the last update exceeds 120 s.

The `ntp_enable` and `kern_enable` are set by the configuration module. Normally, both switches default on, so the daemon can control the time and the kernel discipline can be used, if available. The `pps_update` switch is set by the protocol module when it believes the PPS provider source is legitimate and operating within nominals. The `ntp_control` switch is set during configuration by interrogating the kernel. If both the `kern_enable` and `ntp_control` switches are set, the daemon disciplines the clock via the kernel and the internal daemon discipline is disabled.

The external clock driver controls the system time and clock selection in the following way. Normally, the driver adjusts the kernel time using the `ntp_adjtime()` system call in the same way as the daemon. In the case where the kernel discipline is to be used intact, the clock offset is provided in this call and the loop operates as specified. In the case where the driver steers only the frequency, the offset is specified as zero.

### 6.2 How to Write a Reference Clock Driver

#### 6.2.1 Description

NTP reference clock support maintains the fiction that the clock is actually an ordinary server in the NTP tradition, but operating at a synthetic stratum of zero. The entire suite of algorithms filter the received data and select the best sources to correct the system clock. No packets are exchanged with a reference clock; however, the transmit, receive and packet procedures are replaced with code to simulate them.

The driver assumes three timescales: standard time maintained by a distant laboratory such as USNO or NIST, reference time maintained by the external radio and the system time maintained by NTP. The radio synchronizes reference time via radio, satellite or modem. As the transmission means may not always be reliable, most radios continue to provide clock updates for some time after signal loss using an internal reference oscillator. In such cases the radio may or may not reveal the time since last synchronized or the estimated time error.

All three timescales run only in Coordinated Universal Time (UTC) and are not adjusted for local timezone or standard/daylight time. The local timezone, standard/daylight indicator and year, if provided, are ignored. However, it is important to determine whether a leap second is to be inserted in the UTC timescale in the near future so NTP can insert it in the system timescale at the appropriate epoch.

The interface routines in the `ntp_refclock.c` source file call the following driver routines via a transfer vector:

- **startup**  The association has just been mobilized. The driver may allocate a private structure and open the device(s) required.

- **shutdown**  The association is about to be demobilized. The driver should close all device(s) and free private structures.

- **receive**  A timecode string is ready for retrieval using the `refclock_gtlin()` or `refclock_gtraw()` routines and provide clock updates.

- **poll**  Called at poll timeout, by default 64 s. Ordinarily, the driver will send a poll sequence to the radio as required.

- **timer**  Called once per second. This can be used for housekeeping functions. In the case with pulse-per-second (PPS) signals, this can be used to process the signals and provide clock updates.

The receive routine retrieves a timecode string via serial or parallel port, PPS signal or other means. It decodes the timecode in days, hours, minutes, seconds and nanoseconds and checks for errors. It provides these data along with the on-time timestamp to the `refclock_process` routine, which saves the computed offset in a 60-sample circular
buffer. On occasion, either by timeout, sample count or call to the poll routine, the driver calls `refclock_receive` to process the circular buffer samples and update the system clock.

The best way to understand how the clock drivers work is to study one of the drivers already implemented, such as `refclock_wwvb.c`. The main interface is the `refclockproc` structure, which contains for most drivers the decoded timecode, on-time timestamp, reference timestamp, exception reports and statistics tallies, etc. The support routines are passed a pointer to the `peer` structure, which contains a pointer to the `refclockproc` structure, which in turn contains a pointer to the unit structure, if used. For legacy purposes, a table `typeunit[type][unit]` contains the peer structure pointer for each configured clock type and unit. This structure should not be used for new implementations.

Radio and modem reference clocks by convention have addresses of the form `127.127.t.u`, where `t` is the clock type and `u` in the range 0-3 is used to distinguish multiple instances of clocks of the same type. Most clocks require a serial or parallel port or special bus peripheral. The particular device is normally specified by adding a soft link `/dev/deviceu` to the particular hardware device.

By convention, reference clock drivers are named in the form `refclock_xxxx.c`, where `xxxx` is a unique string. Each driver is assigned a unique type number, long-form driver name, short-form driver name and device name. The existing assignments are in the Reference Clock Drivers page and its dependencies. All drivers supported by the particular hardware and operating system are automatically detected in the autoconfigure phase and conditionally compiled.

### 6.2.2 Conventions, Fudge Factors and Flags

Most drivers support manual or automatic calibration for systematic offset bias using values encoded in the `fudge` configuration command. By convention, the `time1` value defines the calibration offset in seconds. For those drivers that support statistics collection using the `filegen` utility and the `clockstats` file, the `flag4` switch enables the utility.

If the calibration feature has been enabled, the `flag1` switch is set and the PPS signal is actively disciplining the system time, the `time1` value is automatically adjusted to maintain a residual offset of zero. Once its value has stabilized, the value can be inserted in the configuration file and the calibration feature disabled.

### 6.2.3 Files Which Need to be Changed

When a new reference clock driver is installed, the following files need to be edited. Note that changes are also necessary to properly integrate the driver in the configuration and makefile scripts, but these are decidedly beyond the scope of this page.

`.include/ntp.h` The reference clock type defines are used in many places. Each driver is assigned a unique type number. Unused numbers are clearly marked in the list. A unique `REFCLK_xxxx` identification code should be recorded in the list opposite its assigned type number.

`.libntp/clocktypes.c` The `.libntp/clktype` array is used by certain display functions. A unique short-form name of the driver should be entered together with its assigned identification code.

`.ntpd/ntp_control.c` The `clocktypes` array is used for certain control message displays functions. It should be initialized with the reference clock class assigned to the driver, as per the NTP specification RFC 1305. See the `.include/ntp_control.h` header file for the assigned classes.

`.ntpd/refclock_conf.c` This file contains a list of external structure definitions which are conditionally defined. A new set of entries should be installed similar to those already in the table. The `refclock_conf` array is a set of pointers to transfer vectors in the individual drivers. The external name of the transfer vector should be initialized in correspondence with the type number.
### 6.2.4 Interface Routine Overview

**refclock_newpeer** - initialize and start a reference clock. This routine allocates and initializes the interface structure which supports a reference clock in the form of an ordinary NTP peer. A driver-specific support routine completes the initialization, if used. Default peer variables which identify the clock and establish its reference ID and stratum are set here. It returns one if success and zero if the clock address is invalid or already running, insufficient resources are available or the driver declares a bum rap.

**refclock_unpeer** - shut down a clock This routine is used to shut down a clock and return its resources to the system.

**refclock_transmit** - simulate the transmit procedure This routine implements the NTP transmit procedure for a reference clock. This provides a mechanism to call the driver at the NTP poll interval, as well as provides a reachability mechanism to detect a broken radio or other madness.

**refclock_process** - insert a sample in the circular buffer This routine saves the offset computed from the on-time timestamp and the days, hours, minutes, seconds and nanoseconds in the circular buffer. Note that no provision is included for the year, as provided by some (but not all) radio clocks. Ordinarily, the year is implicit in the Unix file system and hardware/software clock support, so this is ordinarily not a problem.

**refclock_receive** - simulate the receive and packet procedures This routine simulates the NTP receive and packet procedures for a reference clock. This provides a mechanism in which the ordinary NTP filter, selection and combining algorithms can be used to suppress misbehaving radios and to mitigate between them when more than one is available for backup.

**refclock_gtraw, refclock_gtlin** - read the buffer and on-time timestamp These routines return the data received from the clock and the on-time timestamp. The refclock_gtraw routine returns a batch of one or more characters returned by the Unix terminal routines in raw mode. The refclock_gtlin routine removes the parity bit and control characters and returns all the characters up to and including the line terminator. Either routine returns the number of characters delivered.

**refclock_open** - open a serial port for reference clock This routine opens a serial port for I/O and sets default options. It returns the file descriptor if success and zero if failure.

**refclock_ioctl** - set serial port control functions This routine attempts to hide the internal, system-specific details of serial ports. It can handle POSIX (termios), SYSV (termio) and BSD (sgtty) interfaces with varying degrees of success. The routine returns one if success and zero if failure.

**refclock_ppsapi** This routine initializes the Pulse-per-Second interface (see below).

**refclock_pps** This routine is called once per second to read the latest PPS offset and save it in the circular buffer (see below).

### 6.2.5 Pulse-per-Second Interface

When the Pulse-per-Second Application Interface ([RFC 2783](https://tools.ietf.org/html/rfc2783)) is present, a compact PPS interface is available to all drivers. See the Mitigation Rules and the Prefer Peer page for further information. To use this interface, include the timeppps.h and refclock_atom.h header files and define the refclock_atom structure in the driver private storage. The timeppps.h file is specific to each operating system and may not be available for some systems.

To use the interface, call refclock_ppsapi from the startup routine passing the device file descriptor and refclock_atom structure pointer. Then, call refclock_pps from the timer routine passing the association pointer and refclock_atom structure pointer. See the refclock_atom.c file for examples and calling sequences. If the PPS signal is valid, the offset sample will be save in the circular buffer and a bit set in the association flags word indicating the sample is valid and the driver an be selected as a PPS peer. If this bit is set when the poll routine is called, the driver calls the refclock_receive routine to process the samples in the circular buffer and update the system clock.
6.3 NTP PARSE clock data formats

The parse driver currently supports several clocks with different query mechanisms. In order for you to find a sample that might be similar to a clock you might want to integrate into parse I’ll sum up the major features of the clocks (this information is distributed in the parse/clk_*.c and ntpd/refclock_parse.c files).

6.3.1 Meinberg clocks

Meinberg: start=<STX>, end=<ETX>, sync on start

```
pattern="\2D: . . ;T: ;U: . . ; \3"
pattern="\2 . . ; ; : : ; \3"
pattern="\2 . . ; ; : : ; : ; ; . . "
```

Meinberg is a German manufacturer of time code receivers. Those clocks have a pretty common output format in the stock version. In order to support NTP Meinberg was so kind to produce some special versions of the firmware for the use with NTP. So, if you are going to use a Meinberg clock please ask whether there is a special Uni Erlangen version. You can reach Meinberg via the Web. Information can also be ordered via eMail from info@meinberg.de

General characteristics:

Meinberg clocks primarily output pulse per second and a describing ASCII string. This string can be produced in two modes: either upon the reception of a question mark or every second. NTP uses the latter mechanism. DCF77 AM clocks have a limited accuracy of a few milliseconds. The DCF77 PZF5xx variants provide higher accuracy and have a pretty good relationship between RS232 time code and the PPS signal. Except for early versions of the old GPS166 receiver type, Meinberg GPS receivers have a very good timing relationship between the datagram and the pulse. The beginning of the start bit of the first character has basically the same accuracy as the PPS signal, plus a jitter of up to 1 bit time depending on the selected baud rate, i.e. 52 \(\mu\)s @ 19200. PPS support should always be used, if possible, in order to yield the highest possible accuracy.

The preferred tty setting for Meinberg DCF77 receivers is 9600/7E2:

```
CFLAG (B9600|CS7|PARENB|CREAD|HUPCL)
IFLAG (IGNBRK|IGNPAR|ISTRIP)
OFLAG 0
LFLAG 0
```

The tty setting for Meinberg GPS16x/17x receivers is 19200/8N1:

```
CFLAG (B19200|CS8|PARENB|CREAD|HUPCL)
IFLAG (IGNBRK|IGNPAR|ISTRIP)
OFLAG 0
LFLAG 0
```

All clocks should be run at datagram once per second.

Format of the Meinberg standard time string:

```
<STX>D:dd.mm.yy;T:w;U:hh.mm.ss;uvxy<ETX>
pos: 0 00000000111111111122222222233 3
     1 234567890123456789012345678901 2
```
### Format of the Uni Erlangen time string for PZF5xx receivers:

```
<STX>dd.mm.yy; w; hh:mm:ss; tuvxyza<ETX>
pos: 0 0000000011111111111122222222223 3
 1 234567890123456789012345678901 2
```

### Format of the Uni Erlangen time string for GPS16x/GPS17x receivers:

```
<STX>dd.mm.yy; w; hh:mm:ss; +uu:uu; uvxyzab; ll.lllln lll.lllle hhhhm<ETX>
pos: 0 0000000011111111111122222222223 3
 1 2345678901234567890123456789012345678901234567890123456789012345678901 6
```

---

88 Chapter 6. Reference Clock Support
Examples for Uni Erlangen strings from GPS receivers:

\x02 09.07.93; 5; 08:48:26; +00:00; ; 49.5736N 11.0280E 373m \x03
\x02 08.11.06; 3; 14:39:39; +00:00; ; 51.9828N 9.2258E 176m \x03

The Uni Erlangen formats should be used preferably. Newer Meinberg GPS receivers can be configured to transmit that format, for older devices there may be a special firmware version available.

For the Meinberg parse look into clk_meinberg.c

### 6.3.2 Raw DCF77 Data via serial line

**RAWDCF:** end=TIMEOUT>1.5s, sync each char (any char), generate pseudo time codes, fixed format

direct DCF77 code input

In Europe it is relatively easy/cheap the receive the german time code transmitter DCF77. The simplest version to process its signal is to feed the 100/200ms pulse of the demodulated AM signal via a level converter to an RS232 port at 50Baud. parse/clk_rawdcf.c holds all necessary decoding logic for the time code which is transmitted each minute for one minute. A bit of the time code is sent once a second.

The preferred tty setting is:

| CFLAG     | (B50|CS8|CREAD|CLOCAL) |
|-----------|------------------|
| IFLAG     | 0                |
| OFLAG     | 0                |
| LFLAG     | 0                |

### 6.3.3 DCF77 raw time code

From “Zur Zeit”, Physikalisch-Technische Bundesanstalt (PTB), Braunschweig und Berlin, März 1989

Timecode transmission:

**AM:**

time marks are send every second except for the second before the next minute mark
time marks consist of a reduction of transmitter power to 25% of the nominal levelthe falling edge is the time indication (on time)
time marks of a 100ms duration constitute a logical 0time marks of a 200ms duration constitute a logical 1

see the spec. (basically a (non-)inverted psuedo random phase shift) encoding:

**FM:**

Second  Contents
0  – 10 AM: free, FM: 0
11 - 14 free
15  R   alternate antenna
16   A1  expect zone change (1 hour before)
17 - 18 Z1,Z2  - time zone
   0 0 illegal
   0 1 MEZ (MET)
   1 0 MESZ (MED, MET DST)
   1 1 illegal
19   A2  expect leap insertion/deletion (1 hour before)
20   S   start of time code (1)
21 - 24 M1  - BCD (lsb first) Minutes
25 - 27 M10  - BCD (lsb first) 10 Minutes
28   P1  - Minute Parity (even)
29 - 32 H1  - BCD (lsb first) Hours
33 - 34 H10  - BCD (lsb first) 10 Hours
35   P2  - Hour Parity (even)
36 - 39 D1  - BCD (lsb first) Days
40 - 41 D10  - BCD (lsb first) 10 Days
42 - 44 DW  - BCD (lsb first) day of week (1: Monday -> 7: Sunday)
45 - 49 M01  - BCD (lsb first) Month
50   M01  - 10 Months
51 - 53 Y1  - BCD (lsb first) Years
54 - 57 Y10  - BCD (lsb first) 10 Years
58   P3  - Date Parity (even)
59   - usually missing (minute indication), except for leap insertion

6.3.4 Schmid clock

Schmid clock: needs poll, binary input, end=`xFF`, sync start

The Schmid clock is a DCF77 receiver that sends a binary time code at the reception of a flag byte. The contents if the flag byte determined the time code format. The binary time code is delimited by the byte 0xFF.

TTY setup is:

```
CFLAG (B1200|CS8|CREAD|CLOCAL)
IFLAG 0
OFLAG 0
LFLAG 0
```

The command to Schmid’s DCF77 clock is a single byte; each bit allows the user to select some part of the time string, as follows (the output for the lsb is sent first).

Bit 0: time in MEZ, 4 bytes *binary, not BCD*; hh:mm:ss.tenths
Bit 1: date 3 bytes *binary, not BCD: dd.mm.yy
Bit 2: week day, 1 byte (unused here)
Bit 3: time zone, 1 byte, 0=MET, 1=MEST. (unused here)
Bit 4: clock status, 1 byte, 0=time invalid,
   1=time from crystal backup,
   3=time from DCF77
Bit 5: transmitter status, 1 byte,
   0: backup antenna
   bit 1: time zone change within 1h
   bit 3,2: TZ 01=MEST, 10=MET
   bit 4: leap second will be
      added within one hour
   bits 5-7: Zero
Bit 6: time in backup mode, units of 5 minutes (unused here)
6.3.5 Trimble SV6 ASCII time code (TAIP)

Trimble SV6: needs poll, ascii timecode, start='>', end='<', query='>QTM<', eol='<'

Trimble SV6 is a GPS receiver with PPS output. It needs to be polled. It also need a special tty mode setup (EOL='<').

TTY setup is:

<table>
<thead>
<tr>
<th>FLAG</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFLAG</td>
<td>(B4800</td>
</tr>
<tr>
<td>IFLAG</td>
<td>(BRKINT</td>
</tr>
<tr>
<td>OFLAG</td>
<td>(OPOST</td>
</tr>
<tr>
<td>LFLAG</td>
<td>(ICANON</td>
</tr>
</tbody>
</table>

Special flags are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARSE_F_PPSPPS</td>
<td>- use CIOGETEV for PPS time stamping</td>
</tr>
<tr>
<td>PARSE_F_PPSONSECOND</td>
<td>- the time code is not related to</td>
</tr>
<tr>
<td></td>
<td>the PPS pulse (so use the time code</td>
</tr>
<tr>
<td></td>
<td>only for the second epoch)</td>
</tr>
</tbody>
</table>

Timecode

0000000000111111111122222222223333333 / char
0123456789012345678901234567890123456 \
>RMhhmmsddddDDMmYYyoonnnvrrrrr;*xx< Actual
----33445566600112222BB7___-____--99- Parse
>RM   1  ;* < Check

6.3.6 ELV DCF7000

ELV DCF7000: end='\r', pattern="--- - - - - - - - \r"

The ELV DCF7000 is a cheap DCF77 receiver sending each second a time code (though not very precise!) delimited by ‘\r’

Timecode

YY-MM-DD-HH-MM-SS-FF\r

FF&0x1 - DST
FF&0x2 - DST switch warning
FF&0x4 - unsynchronised

6.3.7 HOPF 6021 und Kompatible

HOPF Funkuhr 6021 mit serieller Schnittstelle Created by F.Schnekenbuehl <frank@comsys.dofn.de> from clk_rcc8000.c Nortel DASA Network Systems GmbH, Department: ND250 A Joint venture of Daimler-Benz Aerospace and Nortel.

hopf Funkuhr 6021
used with 9600,8N1,
UTC via serial line
"Sekundenvorlauf" ON
ETX zum Sekundenvorlauf ON
dataformat 6021
output time and date
transmit with control characters
transmit every second

Type 6021 Serial Output format
6.3.8 Diem Computime Clock

The Computime receiver sends a datagram in the following format every minute.

```
```

- **T** Startcharacter "T" specifies start of the timestamp
- **YY** Year MM Month 1-12
- **MD** Day of the month
- **WD** Day of week
- **HH** Hour
- **MM** Minute
- **SS** Second
- **CR** Carriage return
- **LF** Linefeed
6.3.9 WHARTON 400A Series Clock with a 404.2 Serial interface

The WHARTON 400A Series clock is able to send date/time serial messages in 7 output formats. We use format 1 here because it is the shortest. We set up the clock to send a datagram every second. For use with this driver, the WHARTON 400A Series clock must be set up as follows:

<table>
<thead>
<tr>
<th>Programmable Option No</th>
<th>Selected Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BST or CET display</td>
<td>3</td>
</tr>
<tr>
<td>No external controller</td>
<td>7</td>
</tr>
<tr>
<td>Serial Output Format 1</td>
<td>9</td>
</tr>
<tr>
<td>Baud rate 9600 bps</td>
<td>10</td>
</tr>
<tr>
<td>Bit length 8 bits</td>
<td>11</td>
</tr>
<tr>
<td>Parity even</td>
<td>12</td>
</tr>
</tbody>
</table>

WHARTON 400A Series output format 1 is as follows:

```
Timestamp STXssmmhhDDMMYYSETX
Pos 0 12345678901234
     0 000000000000011111
STX start transmission (ASCII 0x02)
ETX end transmission (ASCII 0x03)
ss Second expressed in reversed decimal (units then tens)
mm Minute expressed in reversed decimal
hh Hour expressed in reversed decimal
DD Day of month expressed in reversed decimal
MM Month expressed in reversed decimal (January is 1)
YY Year (without century) expressed in reversed decimal
S Status byte : 0x30 +
bit 0 0 = MSF source 1 = DCF source
bit 1 0 = Winter time 1 = Summer time
bit 2 0 = not synchronised 1 = synchronised
bit 3 0 = no early warning 1 = early warning
```

6.4 How to build new PARSE clocks

Prerequisites:

• Does the system you want the clock connect to have the include files termio.h or termios.h? (You need them for the parse driver)

What to do:

Make a conversion module (libparse/clk_*.c)

1. What is the time code format?

   • find year, month, day, hour, minute, second, status (synchronised or not), possibly time zone information (you need to give the offset to UTC) You will have to convert the data from a string into a struct `clocktime`:

   ```c
   struct clocktime /* clock time broken up from time code */
   {
     long day;
     long month;
     long year;
     long hour;
     long minute;
   }
   ```
Conversion is usually simple and straightforward. For the flags following values can be OR’ed together:

- **PARSEB_ANNOUNCE**: switch time zone warning (informational only)
- **PARSEB_POWERUP**: no synchronisation - clock confused (must set then)
- **PARSEB_NOSYNC**: timecode currently not confirmed (must set then) usually on reception error when there is still a chance the generated time is still ok.
- **PARSEB_DST**: DST in effect (informational only)
- **PARSEB_UTC**: timecode contains UTC time (informational only)
- **PARSEB_LEAPADD**: LEAP addition warning (prior to leap happening - must set when imminent) also used for time code that do not encode the direction (as this is currently the default).
- **PARSEB_LEAPDEL**: LEAP deletion warning (prior to leap happening - must set when imminent)
- **PARSEB_ALTERNATE**: backup transmitter (informational only)
- **PARSEB_POSITION**: geographic position available (informational only)
- **PARSEB_LEAPSECOND**: actual leap second (this time code is the leap second - informational only)

These are feature flags denoting items that are supported by the clock:

- **PARSEB_S_LEAP**: supports LEAP - might set PARSEB_LEAP
- **PARSEB_S_ANTENNA**: supports ANTENNA - might set PARSEB_ALTERNATE
- **PARSEB_S_PPS**: supports PPS time stamping
- **PARSEB_S_POSITION**: supports position information (GPS)

If the utctime field is non-zero this value will be take as time code value. This allows for conversion routines that already have the utc time value. The utctime field gives the seconds since Jan 1st 1970, 0:00:00. The useconds field gives the respective usec value. The fields for date and time (down to second resolution) will be ignored.

Conversion is done in the cvt_ routine in parse/clk_* files. look in them for examples. The basic structure is:

```c
struct clockformat <yourclock>_format = {
    lots of fields for you to fill out (see below)
};

static cvt_<yourclock>()
...
{
    if (<I do not recognize my time code>) {
        return CVT_NONE;
    } else {
        if (<conversion into clockformat is ok>) {
            <set all necessary flags>;
            return CVT_OK;
        } else {
            return CVT_FAIL|CVT_BADFMT;
        }
    }
}
```
The struct clockformat is the interface to the rest of the parse driver - it holds all information necessary for finding the clock message and doing the appropriate time stamping.

```c
struct clockformat
{
    u_long (*input)(); /* input routine - your routine - cvt_<yourclock> */
    u_long (*convert)(); /* conversion routine - your routine - cvt_<yourclock> */
    /* routine for handling RS232 sync events (time stamps) - usually sync_simple */
    u_long (*syncpps)(); /* PPS input routine - usually pps_one */
    void *data; /* local parameters - any parameters/data/configuration info your conversion 
                 routine might need */
    char *name; /* clock format name - Name of the time code */
    unsigned short length; /* maximum length of data packet for your clock format */
    u_long flags; /* information for the parser what to look for */
};
```

The above should have given you some hints on how to build a clk_*.c file with the time code conversion. See the examples and pick a clock closest to yours and tweak the code to match your clock.

In order to make your clk_*.c file usable a reference to the clockformat structure must be put into parse_conf.c.

2. TTY setup and initialisation/configuration will be done in ntpd/refclock_parse.c.

   • Find out the exact tty settings for your clock (baud rate, parity, stop bits, character size, ...) and note them in terms of termio*.h c_cflag macros.
   • in ntpd/refclock_parse.c fill out a new the struct clockinfo element (that allocates a new “IP” address - see comments) (see all the other clocks for example)

```c
struct clockinfo
{
    u_long cl_flags; /* operation flags (io modes) */
    PARSE_F_PPSPPS use loopfilter PPS code (CIOGETEV)
    PARSE_F_PPSONSECOND PPS pulses are on second
    usually flags stay 0 as they are used only for special setups
    void (*cl_poll)(); /* active poll routine */
    The routine to call when the clock needs data sent to it in order to 
    get a time code from the clock (e.g. Trimble clock)
    int (*cl_init)(); /* active poll init routine */
    The routine to call for very special initializations.
    void (*cl_event)(); /* special event handling (e.g. reset clock) */
    What to do, when an event happens - used to re-initialize clocks on timeout.
    void (*cl_end)(); /* active poll end routine */
    The routine to call to undo any special initialisation (free memory/timers)
    void *cl_data; /* local data area for "poll" mechanism */
    local data for polling routines
};
```

6.4. How to build new PARSE clocks
NTP, Release 4.2.8p3

```c
struct clockinfo {
    u_fp cl_rootdelay;  /* rootdelay */
    NTP rootdelay estimate (usually 0)
    u_long cl_basedelay;  /* current offset - unsigned l_fp
    fractional part (fraction) by
    which the RS232 time code is
    delayed from the actual time. */
    u_long cl_ppsdelay;  /* current PPS offset - unsigned l_fp fractional
    time (fraction) by which the PPS time stamp is delayed (usually 0)
    part */
    char *cl_id;  /* ID code (usually "DCF") */
    Refclock id - (max 4 chars)
    char *cl_description;  /* device name */
    Name of this device.
    char *cl_format;  /* fixed format */
    If the data format cann not ne detected automatically this is the name
    as in clk_*c clockformat.
    u_char cl_type;  /* clock type (ntp control) */
    Type if clock as in clock status word (ntp control messages) - usually 0
    u_long cl_maxunsync;  /* time to trust oscillator after losing synch */
    seconds a clock can be trusted after losing synchronisation.
    u_long cl_speed;  /* terminal input & output baudrate */
    u_long cl_cflag;  /* terminal io flags */
    u_long cl_iflag;  /* terminal io flags */
    u_long cl_oflag;  /* terminal io flags */
    u_long cl_lflag;  /* terminal io flags */
    termio*.h tty modes.
    u_long cl_samples;  /* samples for median filter */
    u_long cl_keep;  /* samples for median filter to keep */
    median filter parameters - smoothing and rejection of bad samples
};

Well, this is very sketchy, I know. But I hope it helps a little bit. The best way is to look which clock comes closest to
yours and tweak that code.

Two sorts of clocks are used with parse. Clocks that automatically send their time code (once a second) do not need
entries in the poll routines because they send the data all the time. The second sort are the clocks that need a command
sent to them in order to reply with a time code (like the Trimble clock).

For questions: kardel@acm.org.

Please include an exact description on how your clock works. (Initialisation, TTY modes, strings to be sent to it,
responses received from the clock).
7.1 Reference Clock Audio Drivers

7.1.1 Sound Card Drivers

There are some applications in which the computer time can be disciplined to an audio signal, rather than a serial timecode and communications port or special purpose bus peripheral. This is useful in such cases where the audio signal is sent over a telephone circuit, for example, or received directly from a shortwave receiver. In such cases the audio signal can be connected via an ordinary sound card or baseboard audio codec. The suite of NTP reference clock drivers currently includes three drivers suitable for these applications. They include a driver for the Inter Range Instrumentation Group (IRIG) signals produced by many radio clocks and timing devices, another for the Canadian time/frequency radio station CHU and a third for the NIST time/frequency radio stations WWV and WWVH. The radio drivers are designed to work with ordinary inexpensive shortwave radios and may be one of the least expensive ways to build a good primary time server.

All three drivers make ample use of sophisticated digital signal processing algorithms designed to efficiently extract timing signals from noise and interference. The radio station drivers in particular implement optimum linear demodulation and decoding techniques, including maximum-likelihood and soft-decision methods. The documentation page for each driver contains an in-depth discussion on the algorithms and performance expectations. In some cases the algorithms are further analyzed, modeled and evaluated in a technical report.

Currently, the audio drivers work with with Sun operating systems and audio codecs, including SunOS 4.1.3 and Solaris from 2.6 and probably all others in between. They also work with FreeBSD from 4.1 with compatible sound card. In fact, the interface is quite generic and support for other systems, in particular the various Unix generics, should not be difficult. Volunteers are solicited.

The audio drivers include a number of common features designed to groom input signals, suppress spikes and normalize signal levels. An automatic gain control (AGC) feature provides protection against overdriven or underdriven input signals. It is designed to maintain adequate demodulator signal amplitude while avoiding occasional noise spikes. In order to assure reliable operation, the signal level must be in the range where the audio gain control is effective. In general, this means the input signal level must be such as to cause the AGC to set the gain somewhere in the middle of the range from 0 to 255, as indicated in the timecode displayed by the `ntpq` program.

The IRIG and WWV drivers operate by disciplining a logical clock based on the codec sample clock to the audio signal as received. This is done by stuffing or slipping samples as required to maintain exact frequency to the order of 0.1 PPM. In order for the driver to reliably lock on the audio signal, the sample clock frequency tolerance must be less than 250 PPM (.025 percent) for the IRIG driver and half that for the WWV driver. The largest error observed so far is about 60 PPM, but it is possible some sound cards or codecs may exceed that value. In any case, the configuration file command `tinker codec` command can be used to change the systematic offset in units of 125 PPM.

The drivers include provisions to select the input port and to monitor the input signal. The `fudge flag 2` command selects the microphone port if set to zero or the line-in port if set to one. It does not seem useful to specify the compact disc player port. The `fudge flag 3` command enables the input signal monitor using the previously selected
output port and output gain. Both of these flags can be set in the configuration file or remotely using the `ntpd` utility program.

### 7.1.2 Shortwave Radio Drivers

The WWV/H and CHU audio drivers require an external shortwave radio with the radio output - speaker or headphone jack - connected to either the microphone or line-in port on the computer. There is some degree of art in setting up the radio and antenna and getting the setup to work. While the drivers are highly sophisticated and efficient in extracting timing signals from noise and interference, it always helps to have as clear a signal as possible.

The most important factor affecting the radio signal is the antenna. It need not be long - even 15 feet is enough if it is located outside of a metal frame building, preferably on the roof, and away from metallic objects. An ordinary CB whip mounted on a PVC pipe and wooden X-frame on the roof should work well with most portable radios, as they are optimized for small antennas.

The radio need not be located near the computer; in fact, it generally works better if the radio is outside the near field of computers and other electromagnetic noisemakers. It can be in the elevator penthouse connected by house wiring, which can also be used to power the radio. A couple of center-tapped audio transformers will minimize noise pickup and provide phantom power to the radio with return via the building ground.

The WWV/H and CHU transmitters operate on several frequencies simultaneously, so that in most parts of North America at least one frequency supports propagation to the receiver location at any given hour. While both drivers support the ICOM CI-V radio interface and can tune the radio automatically, computer-tunable radios are expensive and probably not cost effective compared to a GPS receiver. So, the radio frequency must usually be fixed and chosen by compromise.

Shortwave (3-30 MHz) radio propagation phenomena are well known to shortwave enthusiasts. The phenomena generally obey the following rules:

- The optimum frequency is higher in daytime than nighttime, stays high longer on summer days and low longer on winter nights.
- Transitions between daytime and nighttime conditions generally occur somewhat after sunrise and sunset at the midpoint of the path from transmitter to receiver.
- Ambient noise (static) on the lower frequencies follows the thunderstorm season, so is higher on summer afternoons and evenings.
- The lower frequency bands are best for shorter distances, while the higher bands are best for longer distances.
- The optimum frequencies are higher at the peak of the 11-year sunspot cycle and lower at the trough. The current sunspot cycle began at the minimum in late 2006 and should reach its peak in 2012.

The best way to choose a frequency is to listen at various times over the day and determine the highest (daytime) and lowest (nighttime) frequencies that work well. Choose the frequency that works for the most number of hours in the day, usually the highest frequency. For instance, on the east coast the best compromise CHU frequency is 7335 kHz and the best WWV frequency is 15 MHz.

### 7.1.3 Autotune Modes

The shortwave drivers include support for an optional autotune function compatible with ICOM receivers and transceivers. The `mode` keyword of the `server` configuration command specifies the ICOM ID select code in decimal. A missing or zero argument disables the CI-V interface. Since all ICOM select codes are less than 128, the high order bit of the code is used by the driver to specify the baud rate. If this bit is not set, the rate is 9600 bps for the newer radios; if set, the rate is 1200 bps for the older radios. Following are the ID select codes for the known radios.
### 7.1.4 Setup and Debugging Aids

The audio drivers include extensive setup and debugging support to help hook up the audio signals and monitor the driver operations. The documentation page for each driver describes the various messages that can be produced either in real time or written to the `clockstats` file for later analysis. Of particular help in verifying signal connections and compatibility is a provision to monitor the signal via headphones or speaker.

Connecting radios and IRIG devices to the computer and verifying correct configuration is somewhat of a black art. The signals have to be connected to the correct ports and the signal level maintained within tolerances. Some radios have recorder outputs which produce a microphone-level signal not affected by the volume control. These signals can be connected to the microphone port on the computer. If the radio does not have a recorder output, connect the headphone or speaker output to the line-in port and adjust the volume control so the driver indicates comfortably above the minimum specified and the AGC level somewhere in the middle of the range 0-255. IRIG signals are usually much larger than radio outputs, usually in the range to several volts and may even overload the line-in port. In such cases the signal is designed to drive a cable terminated with a 50-ohm resistor, which results in a level the line-in port can handle.

It is very easy to underdrive or overdrive the audio codec, in which case the drivers will not synchronize to the signal. The drivers use `fudge flag 2` to enable audio monitoring of the input signal. This is useful during setup to confirm the signal is actually reaching the audio codec and generally free of noise and interference. Note that the monitor volume must be set before the driver is started.

The drivers write a synthesized timecode to the `clockstats` file each time the clock is set or verified and at other times if verbose monitoring is enabled. The format includes several fixed-length fields defining the UTC time to the millisecond, together with additional variable-length fields specific to each driver. The data include the intervals since the clock was last set or verified, the audio gain and various state variables and counters specific to each driver.

### 7.2 IRIG Audio Decoder

#### 7.2.1 Synopsis

**Address:** 127.127.6.u  
**Reference ID:** IRIG  
**Driver ID:** IRIG_AUDIO  
**Audio Device:** `/dev/audio` and `/dev/audioctl`
7.2.2 Description

This driver synchronizes the computer time using the Inter-Range Instrumentation Group (IRIG) standard time distribution signal. This signal is generated by several radio clocks, including those made by Arbiter, Austron, Bancomm, Odetics, Spectracom, Symmetricom and TrueTime, among others, although it is often an add-on option. The signal is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC.

The driver requires an audio codec or sound card with sampling rate 8 kHz and μ-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating systems, including Solaris, FreeBSD and Linux, among others. In this implementation, only one audio driver and codec can be supported on a single machine. In order to assure reliable signal capture, the codec frequency error must be less than 250 PPM (.025 percent). If necessary, the `tinker codec` configuration command can be used to bracket the codec frequency to this range.

For proper operation the IRIG signal source should be configured for analog signal levels, not digital TTL levels. In most radios the IRIG signal is driven ±10 V behind 50 Ohms. In such cases the cable should be terminated at the line-in port with a 50-Ohm resistor to avoid overloading the codec. Where feasible, the IRIG signal source should be operated with signature control so that, if the signal is lost or mutilated, the source produces an unmodulated signal, rather than possibly random digits. The driver automatically rejects the data and declares itself unsynchronized in this case. Some devices, in particular Spectracom radio/satellite clocks, provide additional year and status indication; other devices may not.

In general and without calibration, the driver is accurate within 500 μs relative to the IRIG time. After calibrating relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is less than 20 μs with standard deviation 10 μs. Most of this is due to residuals after filtering and averaging the raw codec samples, which have an inherent jitter of 125 μs. The processor load due to the driver is 0.6 percent on the P4.

However, be acutely aware that the accuracy with Solaris 2.8 and beyond has been seriously degraded to the order of several milliseconds. The Sun kernel driver has a sawtooth modulation with amplitude over 5 ms P-P and period 5.5 s. This distortion is especially prevalent with Sun Blade 1000 and possibly other systems.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.

This driver incorporates several features in common with other audio drivers such as described in the Radio CHU Audio Demodulator/Decoder and the Radio WWV/H Audio Demodulator/Decoder pages. They include automatic gain control (AGC), selectable audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hookup, debugging and monitoring, see the Reference Clock Audio Drivers page.

7.2.3 Technical Overview

The IRIG signal format uses an amplitude-modulated carrier with pulse-width modulated data bits. For IRIG-B, the carrier frequency is 1000 Hz and bit rate 100 b/s; for IRIG-E, the carrier frequency is 100 Hz and bit rate 10 b/s. While IRIG-B provides the best accuracy, generally within a few tens of microseconds relative to IRIG time, it can also generate a significant processor load with older workstations. Generally, the accuracy with IRIG-E is about ten times worse than IRIG-B, but the processor load is somewhat less. Technical details about the IRIG formats can be found in IRIG Standard 200-98.

The driver processes 8000-Hz μ-law companded samples using separate signal filters for IRIG-B and IRIG-E, a comb filter, envelope detector and automatic threshold corrector. An infinite impulse response (IIR) 1000-Hz bandpass filter is used for IRIG-B and an IIR 130-Hz lowpass filter for IRIG-E. These are intended for use with noisy signals, such as might be received over a telephone line or radio circuit, or when interfering signals may be present in the audio passband. The driver determines which IRIG format is in use by sampling the amplitude of each filter output and selecting the one with maximum signal.
Cycle crossings relative to the corrected slice level determine the width of each pulse and its value - zero, one or position identifier (PI). The data encode ten characters (20 BCD digits) which determine the second, minute, hour and day of the year and with some IRIG generators the year and synchronization condition. The comb filter exponentially averages the corresponding samples of successive baud intervals in order to reliably identify the reference carrier cycle. A type-II phase-lock loop (PLL) performs additional integration and interpolation to accurately determine the zero crossing of that cycle, which determines the reference timestamp. A pulse-width discriminator demodulates the data pulses, which are then encoded as the BCD digits of the timecode. The timecode and reference timestamp are updated once each second with IRIG-B (ten seconds with IRIG-E) and local clock offset samples saved for later processing. At poll intervals of 64 s, the saved samples are processed by a median filter and used to update the system clock.

7.2.4 Monitor Data

The timecode format used for debugging and data recording includes data helpful in diagnosing problems with the IRIG signal and codec connections. The driver produces one line for each timecode in the following format:

```
00 00 98 23 19:26:52 2782 143 0.694 10 0.3 66.5 3094572411.00027
```

If clockstats is enabled, the most recent line is written to the clockstats file every 64 s. If verbose recording is enabled (fudge flag 4) each line is written as generated.

The first field contains the error flags in hex, where the hex bits are interpreted as below. This is followed by the year of century, day of year and time of day. Note that the time of day is for the previous minute, not the current time. The status indicator and year are not produced by some IRIG devices and appear as zeros. Following these fields are the carrier amplitude (0-3000), codec gain (0-255), modulation index (0-1), time constant (4-10), carrier phase error (0 ± 0.5) and carrier frequency error (PPM). The last field is the on-time timestamp in NTP format.

The error flags are defined as follows in hex:

- **x01** Low signal. The carrier amplitude is less than 100 units. This is usually the result of no signal or wrong input port.
- **x02** Frequency error. The codec frequency error is greater than 250 PPM. This may be due to wrong signal format or (rarely) defective codec.
- **x04** Modulation error. The IRIG modulation index is less than 0.5. This is usually the result of an overdriven codec, wrong signal format or wrong input port.
- **x08** Frame synch error. The decoder frame does not match the IRIG frame. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal. It may also be the result of an IRIG signature check which indicates a failure of the IRIG signal synchronization source.
- **x10** Data bit error. The data bit length is out of tolerance. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal.
- **x20** Seconds numbering discrepancy. The decoder second does not match the IRIG second. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal.
- **x40** Codec error (overrun). The machine is not fast enough to keep up with the codec.
- **x80** Device status error (Spectracom).

7.2.5 Fudge Factors

```
time1 time
```
Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

```
time2 time
```
Not used by this driver.

```
stratum number
```
Specifies the driver stratum, in decimal from 0 to 15, with default 0.
refid string  Specifies the driver reference identifier, an ASCII string from one to four characters, with default IRIG.

flag1 0 | 1 Not used by this driver.

flag2 0 | 1 Specifies the microphone port if set to zero or the line-in port if set to one. It does not seem useful to specify the compact disc player port.

flag3 0 | 1 Enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.

flag4 0 | 1 Enable verbose clockstats recording if set.

7.3 Radio CHU Audio Demodulator/Decoder

7.3.1 Synopsis

Address: 127.127.7.u
Reference ID: CHU
Driver ID: CHU
Modem Port: /dev/chuu; 300 baud, 8-bits, no parity
Autotune Port: /dev/icom; 1200/9600 baud, 8-bits, no parity
Audio Device: /dev/audio and /dev/audioctl

7.3.2 Description

This driver synchronizes the computer time using shortwave radio transmissions from Canadian time/frequency station CHU in Ottawa, Ontario. CHU transmissions are made continuously on 3.330, 7.850 and 14.670 MHz in upper sideband, compatible AM mode. An ordinary shortwave receiver can be tuned manually to one of these frequencies or, in the case of ICOM receivers, the receiver can be tuned automatically as propagation conditions change throughout the day and season.

The driver can be compiled to use either an audio codec or soundcard, or a Bell 103-compatible, 300-b/s modem or modem chip, as described on the Pulse-per-second (PPS) Signal Interfacing page. If compiled for a modem, the driver uses it to receive the radio signal and demodulate the data. If compiled for the audio codec, it requires a sampling rate of 8 kHz and μ-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating systems, including Solaris, FreeBSD and Linux, among others. The radio is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC. In this implementation, only one audio driver and codec can be supported on a single machine.

In general and without calibration, the driver is accurate within 1 ms relative to the broadcast time when tracking a station. However, variations up to 0.3 ms can be expected due to diurnal variations in ionospheric layer height and ray geometry. In Newark DE, 625 km from the transmitter, the predicted one-hop propagation delay varies from 2.8 ms in sunlight to 2.6 ms in moonlight. When not tracking the station the accuracy depends on the computer clock oscillator stability, ordinarily better than 0.5 PPM.

After calibration relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is generally within 0.2 ms short-term with 0.4 ms jitter. The long-term mean offset varies up to 0.3 ms due to propagation path geometry variations. The processor load due to the driver is 0.4 percent on the P4.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.
This driver incorporates several features in common with other audio drivers such as described in the Radio WWV/H Audio Demodulator/Decoder and the IRIG Audio Decoder pages. They include automatic gain control (AGC), selectable audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hookup, debugging and monitoring, see the Reference Clock Audio Drivers page.

7.3.3 Technical Overview

The driver processes 8-kHz $\mu$-law companded codec samples using maximum-likelihood techniques which exploit the considerable degree of redundancy available in each broadcast message or burst. As described below, every character is sent twice and, in the case of format A bursts, the burst is sent eight times every minute. The single format B burst is considered correct only if every character matches its repetition in the burst. For the eight format A bursts, a majority decoder requires more than half of the 16 repetitions for each digit decode to the same value. Every character in every burst provides an independent timestamp upon arrival with a potential total of 60 timestamps for each minute.

The CHU timecode format is described on the CHU website. A timecode is assembled when all bursts have been received in each minute. The timecode is considered valid and the clock set when at least one valid format B burst has been decoded and the majority decoder declares success. Once the driver has synchronized for the first time, it will appear reachable and selectable to discipline the system clock. It is normal on occasion to miss a minute or two due to signal fades or noise. If eight successive minutes are missed, the driver is considered unreachable and the system clock will free-wheel at the latest determined frequency offset. Since the signals are almost always available during some period of the day and the NTP clock discipline algorithms are designed to work well even with long intervals between updates, it is unlikely that the system clock will drift more than a few milliseconds during periods of signal loss.

7.3.4 Baseband Signal Processing

The program consists of four major parts: the DSP modem, maximum-likelihood UART, burst assembler and majority decoder. The DSP modem demodulates Bell 103 modem answer-frequency signals; that is, frequency-shift keyed (FSK) tones of 2225 Hz (mark) and 2025 Hz (space). It consists of a 500-Hz bandpass filter centered on 2125 Hz followed by a limiter/discriminator and raised-cosine lowpass filter optimized for the 300-b/s data rate.

The maximum likelihood UART is implemented using a set of eight 11-stage shift registers, one for each of eight phases of the 300-b/s bit clock. At each phase a new baseband signal from the DSP modem is shifted into the corresponding register and the maximum and minimum over all 11 samples computed. This establishes a span (difference) and slice level (average) over all 11 stages. For each stage, a signal level above the slice is a mark (1) and below that is a space (0). A quality metric is calculated for each register with respect to the slice level and the a-priori signal consisting of a start bit (space), eight arbitrary information bits and two stop bits (mark).

The shift registers are processed in round-robin order as the phases of each bit arrive. At the end of each bit all eight phases are searched for valid framing bits, sufficient span and best metric. The best candidate found in this way represents the maximum-likelihood character. The process then continues for all ten characters in the burst.

The burst assembler processes characters either from the maximum-likelihood UART or directly from the serial port as configured. A burst begins when a character is received and is processed after a timeout interval when no characters are received. If the interval between characters is greater than two characters, but less than the timeout interval, the burst is rejected as a runt and a new burst begun. As each character is received, a timestamp is captured and saved for later processing.

A valid burst consists of ten characters in two replicated five-character blocks, each block representing ten 4-bit BCD digits. The format B blocks sent in second 31 contain the year and other information in ten digits. The eight format A blocks sent in seconds 32-39 contain the timecode in ten digits, the first of which is a framing code (6). The burst assembler must deal with cases where the first character of a format A burst is lost or is noise. This is done using the framing codes to correct the discrepancy, either one character early or one character late.
The burst distance is incremented by one for each bit in the first block that matches the corresponding bit in the second block and decremented by one otherwise. In a format B burst the second block is bit-inverted relative to the first, so a perfect burst of five 8-bit characters has distance -40. In a format A burst the two blocks are identical, so a perfect burst has distance +40. Format B bursts must be perfect to be acceptable; however, format A bursts, which are further processed by the majority decoder, are acceptable if the distance is at least 28.

### 7.3.5 Majority Decoder

Each minute of transmission includes eight format A bursts containing two timecodes for each second from 32 through 39. The majority decoder uses a decoding matrix of ten rows, one for each digit position in the timecode, and 16 columns, one for each 4-bit code combination that might be decoded at that position. In order to use the character timestamps, it is necessary to reliably determine the second number of each burst. In a valid burst, the last digit of the two timecodes in the burst must match and the value must be in the range 2-9 and greater than in the previous burst. As each digit of a valid burst is processed, the value at the row corresponding to the digit position in the timecode and column corresponding to the code found at that position is incremented. At the end of the minute, each row of the decoding matrix encodes the number of occurrences of each code found at the corresponding position.

The maximum over all occurrences at each digit position is the distance for that position and the corresponding code is the maximum-likelihood digit. If the distance is not more than half the total number of occurrences, the decoder assumes a soft error and discards all information collected during the minute. The decoding distance is defined as the sum of the distances over the first nine digits; the tenth digit varies over the seconds and is uncounted.

The result of the majority decoder is a nine-digit timecode representing the maximum-likelihood candidate for the transmitted timecode in that minute. Note that the second and fraction within the minute are always zero and that the actual reference point to calculate timestamp offsets is backdated to the first second of the minute. At this point the timecode block is reformatted and the year, days, hours and minutes extracted along with other information from the format B burst, including DST state, DUT1 correction and leap warning. The reformatting operation checks the timecode for invalid code combinations that might have been left by the majority decoder and rejects the entire timecode if found.

If the timecode is valid, it is passed to the reference clock interface along with the backdated timestamps accumulated over the minute. A perfect set of eight bursts could generate as many as 80 timestamps, but the maximum the interface can handle is 60. These are processed using a median filter and trimmed-mean average, so the resulting system clock correction is usually much better than would otherwise be the case with radio noise, UART jitter and occasional burst errors.

### 7.3.6 Autotune

The driver includes provisions to automatically tune the radio in response to changing radio propagation conditions throughout the day and night. The radio interface is compatible with the ICOM CI-V standard, which is a bidirectional serial bus operating at TTL levels. The bus can be connected to a standard serial port using a level converter such as the CT-17. Further details are on the Reference Clock Audio Drivers page.

If specified, the driver will attempt to open the device `/dev/icom` and, if successful will tune the radio to 3.331 MHz. The 1-kHz offset is useful with a narrowband SSB filter where the passband includes the carrier and modem signals. However, the driver is liberal in what it assumes of the configuration. If the `/dev/icom` link is not present or the open fails or the CI-V bus is inoperative, the driver continues in single-frequency mode.

As long as no bursts are received, the driver cycles over the three frequencies in turn, one minute for each station. When bursts are received from one or more stations, the driver operates in a five-minute cycle. During the first four minutes it tunes to the station with the highest metric. During the last minute it alternates between the other two stations in turn in order to measure the metric.
7.3.7 Debugging Aids

The most convenient way to track the program status is using the ntpq program and the clockvar command. This displays the last determined timecode and related status and error counters, even when the program is not discipline the system clock. If the debugging trace feature (-d on the ntpd command line) is enabled, the program produces detailed status messages as it operates. If the fudge flag 4 is set, these messages are written to the clockstats file. All messages produced by this driver have the prefix chu for convenient filtering with the Unix grep command.

With debugging enabled the driver produces messages in the following formats: A single message beginning with chuB is produced for each format B burst received in second 31, while eight messages beginning with chuA are produced for each format A burst received in seconds 32 through 39 of the minute. The first four fields are

\[
\text{stat sig n b}
\]

where \(\text{stat}\) is the status code, \(\text{sig}\) the character span, \(n\) the number of characters in the burst (9-11) and \(b\) the burst distance (0-40). Good bursts will have spans of a 800 or more and the other numbers near the top of the range specified. See the source for the interpretation of the remaining data in the burst. Note that each character of the burst is encoded as two digits in nibble-swapped order.

If the CI-V interface for ICOM radios is active, a debug level greater than 1 will produce a trace of the CI-V command and response messages. Interpretation of these messages requires knowledge of the CI-V protocol, which is beyond the scope of this document.

7.3.8 Monitor Data

When enabled by the filegen facility, every received timecode is written to the clockstats file in the following format:

\[
\text{sq yyyy ddd hh:mm:ss lw dst du lset agc rfrq bcnt dist tsmp}
\]

- \(s\)  sync indicator
- \(q\)  quality character
- yyyy Gregorian year
- ddd day of year
- hh hour of day
- mm minute of hour
- ss second of minute
- lw leap second warning
- dst DST state
- dut DUT sign and magnitude in deciseconds
- lset minutes since last set
- agc audio gain (0-255)
- ident CHU identifier code
- dist decoder distance
- tsmp timestamps captured

The fields beginning with year and extending through dut are decoded from the received data and are in fixed-length format. The agc and lset fields, as well as the following driver-dependent fields, are in variable-length format.

\(s\) The sync indicator is initially ? before the clock is set, but turns to space when the clock has been correctly set.

\(q\) The quality character is a four-bit hexadecimal code showing which alarms have been raised during the most recent minute. Each bit is associated with a specific alarm condition according to the following:

- 8 Timestamp alarm. Fewer than 20 timestamps have been determined.
- 4 Decoder alarm. A majority of repetitions for at least one digit of the timecode fails to agree.
- 2 Format alarm. One or more bursts contained invalid data or was improperly formatted.
1 Frame alarm. One or more bursts was improperly framed or contained too many repetition errors.

The timestamp and decoder alarms are fatal; the data accumulated during the minute are not used to set the clock. The format and fram alarm are nonfatal; only the data in the burst are discarded.

yyyy ddd hh:mm:ss The timecode format itself is self explanatory. Note that the Gregorian year is decoded directly from the transmitted timecode.

lw The leap second warning is normally space, but changes to L if a leap second is to occur at the end of the month.

dst The DST code for Canada encodes the state for all provinces. It is encoded as two hex characters.

dut The DUT sign and magnitude shows the current UT1 offset relative to the displayed UTC time, in deciseconds. It is encoded as one digit preceded by sign.

lset Before the clock is set, this is the number of minutes since the program was started; after the clock is set, this is the number of minutes since the time was last verified relative to the broadcast signal.

agc The audio gain shows the current codec gain setting in the range 0 to 255. Ordinarily, the receiver audio gain control should be set for a value midway in this range.

ident The CHU identifier CHU followed by the current radio frequency code, if the CI-V interface is active, or CHU if not. The radio frequency is encoded as 0 for 3.330 MHz, 1 for 7.850 MHz and 2 for 14.670 MHz.

dist The decoding distance determined during the most recent minute bursts were received. The values range from 0 to 160, with the higher values indicating better signals. The decoding algorithms require the distance at least 50; otherwise all data in the minute are discarded.

tmp The number of timestamps determined during the most recent minute bursts were received. The values range from 0 to 60, with the higher values indicating better signals. The decoding algorithms require at least 20 timestamps in the minute; otherwise all data in the minute are discarded.

### 7.3.9 Fudge Factors

timel time Specifies the propagation delay for CHU (45:18N 75:45N), in seconds and fraction, with default 0.0.

time2 time Not used by this driver.

stratum number Specifies the driver stratum, in decimal from 0 to 15, with default 0.

refid string Specifies the driver reference identifier, an ASCII string from one to four characters, with default CHU.

flag1 0 | 1 Not used by this driver.

flag2 0 | 1 When the audio driver is compiled, this flag selects the audio input port, where 0 is the mike port (default) and 1 is the line-in port. It does not seem useful to select the compact disc player port.

flag3 0 | 1 When the audio driver is compiled, this flag enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.

flag4 0 | 1 Enable verbose clockstats recording if set.

### 7.4 Radio WWV/H Audio Demodulator/Decoder

#### 7.4.1 Synopsis

Address: 127.127.36.u

Reference ID: WVf or WHf
Driver ID: WWV_AUDIO
Autotune Port: /dev/icom; 1200/9600 baud, 8-bits, no parity
Audio Device: /dev/audio and /dev/audioctl

7.4.2 Description

This driver synchronizes the computer time using shortwave radio transmissions from NIST time/frequency stations WWV in Ft. Collins, CO, and WWVH in Kauai, HI. Transmissions are made continuously on 2.5, 5, 10 and 15 MHz from both stations and on 20 MHz from WWV. An ordinary shortwave receiver can be tuned manually to one of these frequencies or, in the case of ICOM receivers, the receiver can be tuned automatically by the driver as propagation conditions change throughout the day and season. The radio is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC.

The driver requires an audio codec or sound card with sampling rate 8 kHz and μ-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating systems, including Solaris, FreeBSD and Linux, among others. In this implementation only one audio driver and codec can be supported on a single machine. In order to assure reliable signal capture, the codec frequency error must be less than 187 PPM (.0187 percent). If necessary, the tinker codec configuration command can be used to bracket the codec frequency to this range.

In general and without calibration, the driver is accurate within 1 ms relative to the broadcast time when tracking a station. However, variations up to 0.3 ms can be expected due to diurnal variations in ionospheric layer height and ray geometry. In Newark DE, 2479 km from the transmitter, the predicted two-hop propagation delay varies from 9.3 ms in sunlight to 9.0 ms in moonlight. When not tracking the station the accuracy depends on the computer clock oscillator stability, ordinarily better than 0.5 PPM.

After calibration relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is generally within 0.1 ms short-term with 0.4 ms jitter. The long-term mean offset varies up to 0.3 ms due to propagation path geometry variations. The processor load due to the driver is 0.4 percent on the P4.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.

This driver incorporates several features in common with other audio drivers such as described in the Radio CHU Audio Demodulator/Decoder and the IRIG Audio Decoder pages. They include automatic gain control (AGC), selectable audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hookup, debugging and monitoring, see the Reference Clock Audio Drivers page.

7.4.3 Technical Overview

The driver processes 8-kHz μ-law companded codec samples using maximum-likelihood techniques which exploit the considerable degree of redundancy available in the broadcast signal. The WWV signal format is described in NIST Special Publication 432 (Revised 1990) and also available on the WWV/H web site. It consists of three elements, a 5-ms, 1000-Hz pulse, which occurs at the beginning of each second, a 800-ms, 1000-Hz pulse, which occurs at the beginning of each minute, and a pulse-width modulated 100-Hz subcarrier for the data bits, one bit per second. The WWVH format is identical, except that the 1000-Hz pulses are sent at 1200 Hz. Each minute encodes nine BCD digits for the year of century plus seven bits for the daylight savings time (DST) indicator, leap warning indicator and DUT1 correction.

The demodulation and decoding algorithms used by this driver are based on a machine language program developed for the TAPR DSP93 DSP unit, which uses the TI 320C25 DSP chip. The analysis, design and performance of the program for this unit is described in: Mills, D.L. A precision radio clock for WWV transmissions. Electrical Engineering Report 97-8-1, University of Delaware, August 1997, 25 pp. Available from http://www.eecis.udel.edu/~mills/papers.html. For use in this driver, the original program was rebuilt in the C language and adapted to the NTP driver interface. The
algorithms have been modified to improve performance, especially under weak signal conditions and to provide an automatic frequency and station selection feature.

As in the original program, the clock discipline is modelled as a Markov process, with probabilistic state transitions corresponding to a conventional clock and the probabilities of received decimal digits. The result is a performance level with very high accuracy and reliability, even under conditions when the minute beep of the signal, normally its most prominent feature, can barely be detected by ear using a communications receiver.

7.4.4 Baseband Signal Processing

The 1000/1200-Hz pulses and 100-Hz subcarrier are first separated using a 600-Hz bandpass filter centered on 1100 Hz and a 150-Hz lowpass filter. The minute pulse is extracted using an 800-ms synchronous matched filter and pulse grooming logic which discriminates between WWV and WWVH signals and noise. The second pulse is extracted using a 5-ms FIR matched filter for each station and a single 8000-stage comb filter.

The phase of the 100-Hz subcarrier relative to the second pulse is fixed at the transmitter; however, the audio stage in many radios affects the phase response at 100 Hz in unpredictable ways. The driver adjusts for each radio using two 170-ms synchronous matched filters. The I (in-phase) filter is used to demodulate the subcarrier envelope, while the Q (quadrature-phase) filter is used in a type-1 phase-lock loop (PLL) to discipline the demodulator phase.

A bipolar data signal is determined from the matched filter subcarrier envelope using a pulse-width discriminator. The discriminator samples the I channel at 15 ms ($n$), 200 ms ($s_0$) and 500 ms ($s_1$), and the envelope (RMS I and Q channels) at 200 ms ($e_1$) and the end of the second ($e_0$). The bipolar data signal is expressed $2s_1 - s_0 - n$, where positive values correspond to data 1 and negative values correspond to data 0. Note that, since the signals $s_0$ and $s_1$ include the noise $n$, the noise component cancels out. The data bit SNR is calculated as $20 \log_{10}(e_1 / e_0)$. If the driver has not synchronized to the minute pulse, or if the data bit amplitude $e_1$ or SNR are below thresholds, the bit is considered invalid and the bipolar signal is forced to zero.

The bipolar signal is exponentially averaged in a set of 60 accumulators, one for each second, to determine the semi-static miscellaneous bits, such as DST indicator, leap second warning and DUT1 correction. In this design a data average value larger than a positive threshold is interpreted as +1 (hit) and a value smaller than a negative threshold as a -1 (miss). Values between the two thresholds, which can occur due to signal fades, are interpreted as an erasure and result in no change of indication.

7.4.5 Maximum-Likelihood Decoder

The BCD digit in each digit position of the timecode is represented as four data bits. The bits are correlated with the bits corresponding to each of the valid decimal digits in this position. If any of the four bits are invalid, the correlated value for all digits in this position is assumed zero. In either case, the values for all digits are exponentially averaged in a likelihood vector associated with this position. The digit associated with the maximum over all averaged values then becomes the maximum-likelihood candidate for this position and the ratio of the maximum over the next lower value represents the digit SNR.

The decoding matrix contains nine row vectors, one for each digit position. Each row vector includes the maximum-likelihood digit, likelihood vector and other related data. The maximum-likelihood digit for each of the nine digit positions becomes the maximum-likelihood time of the century. A built-in transition function implements a conventional clock with decimal digits that count the minutes, hours, days and years, as corrected for leap seconds and leap years. The counting operation also rotates the likelihood vector corresponding to each digit as it advances. Thus, once the clock is set, each clock digit should correspond to the maximum-likelihood digit as transmitted.

Each row of the decoding matrix also includes a compare counter and the most recently determined maximum-likelihood digit. If a digit likelihood exceeds the decision level and compares with previous digits for a number of successive minutes in any row, the maximum-likelihood digit replaces the clock digit in that row. When this condition is true for all rows and the second epoch has been reliably determined, the clock is set (or verified if it has already
been set) and delivers correct time to the integral second. The fraction within the second is derived from the logical master clock, which runs at 8000 Hz and drives all system timing functions.

### 7.4.6 Master Clock Discipline

The logical master clock is derived from the audio codec clock. Its frequency is disciplined by a frequency-lock loop (FLL) which operates independently of the data recovery functions. The maximum value of the 5-ms pulse after the comb filter represents the on-time epoch of the second. At averaging intervals determined by the measured jitter, the frequency error is calculated as the difference between the epoches over the interval divided by the interval itself. The sample clock frequency is then corrected by this amount divided by a time constant of 8.

When first started, the frequency averaging interval is 8 seconds, in order to compensate for intrinsic codec clock frequency offsets up to 125 PPM. Under most conditions, the averaging interval doubles in stages from the initial value to 1024 s, which results in an ultimate frequency resolution of 0.125 PPM, or about 11 ms/day.

The data demodulation functions operate using the subcarrier clock, which is independent of the epoch. However, the data decoding functions are driven by the epoch. The decoder is phase-locked to the epoch in such a way that, when the clock state machine has reliably decoded the broadcast time to the second, the epoch timestamp of that second becomes a candidate to set the system clock.

The comb filter can have a long memory and is vulnerable to noise and stale data, especially when coming up after a long fade. Therefore, a candidate is considered valid only if the 5-ms signal amplitude and SNR are above thresholds. In addition, the system clock is not set until after one complete averaging interval has passed with valid candidates.

### 7.4.7 Station Identification

It is important that the logical clock frequency is stable and accurately determined, since in many applications the shortwave radio will be tuned to a fixed frequency where WWV or WWVH signals are not available throughout the day. In addition, in some parts of the US, especially on the west coast, signals from either or both WWV and WWVH may be available at different times or even at the same time. Since the propagation times from either station are almost always different, each station must be reliably identified before attempting to set the clock.

Reliable station identification requires accurate discrimination between very weak signals in noise and noise alone. The driver very aggressively soaks up every scrap of signal information, but has to be careful to avoid making pseudo-sense of noise alone. The signal quality metric depends on the minute pulse amplitude and SNR measured in second 0 of the minute, together with the data subcarrier amplitude and SNR measured in second 1. If all four values are above defined thresholds a hit is declared, otherwise a miss. In principle, the data pulse in second 58 is usable, but the AGC in most radios is not fast enough for a reliable measurement.

The number of hits declared in the last 6 minutes for each station represents the high order bits of the metric, while the current minute pulse amplitude represents the low order bits. Only if the metric is above a defined threshold is the station signal considered acceptable. The metric is also used by the autotune function described below and reported in the timecode string.

### 7.4.8 Performance

It is the intent of the design that the accuracy and stability of the indicated time be limited only by the characteristics of the ionospheric propagation medium. Conventional wisdom is that manual synchronization via oscilloscope and HF medium is good only to a millisecond under the best propagation conditions. The performance of the NTP daemon disciplined by this driver is clearly better than this, even under marginal conditions.

The figure below shows the measured offsets over a typical day near the bottom of the sunspot cycle ending in October, 2006. Variations up to ±0.4 ms can be expected due to changing ionospheric layer height and ray geometry over the day and night.
The figure was constructed using a 2.4-GHz P4 running FreeBSD 6.1. For these measurements the computer clock was disciplined within a few microseconds of UTC using a PPS signal and GPS receiver and the measured offsets determined from the filegen peerstats data.

The predicted propagation delay from the WWV transmitter at Boulder, CO, to the receiver at Newark, DE, varies over 9.0-9.3 ms. In addition, the receiver contributes 4.7 ms and the 600-Hz bandpass filter 0.9 ms. With these values, the mean error is less than 0.1 ms and varies $\pm 0.3$ ms over the day as the result of changing ionospheric height and ray geometry.

### 7.4.9 Program Operation

The driver begins operation immediately upon startup. It first searches for one or both of the stations WWV and WWVH and attempts to acquire minute synch. This may take some fits and starts, as the driver expects to see several consecutive minutes with good signals and low jitter. If the autotune function is active, the driver will rotate over all five frequencies and both WWV and WWVH stations until finding a station and frequency with acceptable metric.

While this is going on the the driver acquires second synch, which can take up to several minutes, depending on signal quality. When minute synch has been acquired, the driver accumulates likelihood values for the unit (seconds) digit of the nine timecode digits, plus the seven miscellaneous bits included in the WWV/H transmission format. When a good unit digit has been found, the driver accumulated likelihood values for the remaining eight digits of the timecode. When three repetitions of all nine digits have decoded correctly, which normally takes 15 minutes with good signals, and up to 40 minutes when buried in noise, and the second synch has been acquired, the clock is set (or verified) and is selectable to discipline the system clock.

Once the clock is set, it continues to provide correct timecodes as long as the signal metric is above threshold, as described in the previous section. As long as the clock is correctly set or verified, the system clock offsets are provided once each minute to the reference clock interface, where they are processed using the same algorithms as with other reference clocks and remote servers.

It may happen as the hours progress around the clock that WWV and WWVH signals may appear alone, together or not at all. When the driver has mitigated which station and frequency is best, it sets the reference identifier to the string WWf for WWV and WHf for WWVH, where $f$ is the frequency in megahertz. If the propagation delays have
been properly set with the `fudge time1` (WWV) and `fudge time2` (WWVH) commands in the configuration file, handover from one station to the other is seamless.

Operation continues as long as the signal metric from at least one station on at least one frequency is acceptable. A consequence of this design is that, once the clock is set, the time and frequency are disciplined only by the second synch pulse and the clock digits themselves are driven by the clock state machine. If for some reason the state machine drifts to the wrong second, it would never resynchronize. To protect against this most unlikely situation, if after two days with no signals, the clock is considered unset and resumes the synchronization procedure from the beginning.

Once the system clock been set correctly it will continue to read correctly even during the holdover interval, but with increasing dispersion. Assuming the system clock frequency can be disciplined within 1 PPM, it can coast without signals for several days without exceeding the NTP step threshold of 128 ms. During such periods the root distance increases at 15 μs per second, which makes the driver appear less likely for selection as time goes on. Eventually, when the distance due all causes exceeds 1 s, it is no longer suitable for synchronization. Ordinarily, this happens after about 18 hours with no signals. The `tinker maxdist` configuration command can be used to change this value.

7.4.10 Autotune

The driver includes provisions to automatically tune the radio in response to changing radio propagation conditions throughout the day and night. The radio interface is compatible with the ICOM CI-V standard, which is a bidirectional serial bus operating at TTL levels. The bus can be connected to a standard serial port using a level converter such as the CT-17. Further details are on the Reference Clock Audio Drivers page.

If specified, the driver will attempt to open the device `/dev/icom` and, if successful will activate the autotune function and tune the radio to each operating frequency in turn while attempting to acquire minute synch from either WWV or WWVH. However, the driver is liberal in what it assumes of the configuration. If the `/dev/icom` link is not present or the open fails or the CI-V bus is inoperative, the driver quietly gives up with no harm done.

Once acquiring minute synch, the driver operates as described above to set the clock. However, during seconds 59, 0 and 1 of each minute it tunes the radio to one of the five broadcast frequencies to measure the signal metric as described above. Each of the five frequencies are probed in a five-minute rotation to build a database of current propagation conditions for all signals that can be heard at the time. At the end of each probe a mitigation procedure scans the database and retunes the radio to the best frequency and station found. For this to work well, the radio should be set for a fast AGC recovery time. This is most important while tracking a strong signal, which is normally the case, and then probing another frequency, which may have much weaker signals.

The mitigation procedure selects the frequency and station with the highest valid metric, ties going first to the highest frequency and then to WWV in order. A station is considered valid only if the metric is above a specified threshold; if no station is above the metric, the rotating probes continue until a valid station is found.

The behavior of the autotune function over a typical day is shown in the figure below.
As expected, the lower frequencies prevail when the ray path is in moonlight (0100-1300 UTC) and the higher frequencies when the path is in sunlight (1300-0100 UTC). Note three periods in the figure show zero frequency when signals are below the minimum for all frequencies and stations.

## 7.4.11 Debugging Aids

The most convenient way to track the driver status is using the `ntpq` program and the `clockvar` command. This displays the last determined timecode and related status and error counters, even when the driver is not disciplining the system clock. If the debugging trace feature (`-d` on the `ntpd` command line) is enabled, the driver produces detailed status messages as it operates. If the `fudge flag 4` is set, these messages are written to the `clockstats` file. All messages produced by this driver have the prefix `wwv` for convenient filtering with the Unix `grep` command.

The autotune process produces diagnostic information along with the timecode. This is very useful for evaluating the performance of the algorithms, as well as radio propagation conditions in general. The message is produced once each minute for each frequency in turn after minute synch has been acquired.

```
wwv5 status agc epoch secamp/secsnr datamp/datsnr wwv wwvh
```

where the fields after the `wwv5` identifier are: `status` contains status bits, `agc` audio gain, `epoch` second epoch, `secamp/secsnr` second pulse amplitude/SNR, and `wwv` and `wwvh` are two sets of fields, one each for WWV and WWVH. Each of the two fields has the format

```
ident score metric minamp/minsnr
```

where `ident` encodes the station (`WV` for WWV, `WH` for WWVH) and frequency (2, 5, 10, 15 or 20), `score` 32-bit shift register recording the hits (1) and misses (0) of the last 32 probes (hits and misses enter from the right), `metric` is described above, and `minamp/minsnr` is the minute pulse amplitude/SNR. An example is:

```
wwv5 000d 111 5753 3967/20.1 3523/10.2 WV20 bdeff 100 8348/30.0 WH20 0000 1 22/-12.4
```

There are several other messages that can occur; these are documented in the source listing.
7.4.12 Monitor Data

When enabled by the filegen facility, every received timecode is written to the clockstats file in the following format:

```
sq yyyy ddd hh:mm:ss l d du lset agc ident metric errs freq avg
```

The fields beginning with `yyyy` and extending through `du` are decoded from the received data and are in fixed-length format. The remaining fields are in variable-length format. The fields are as follows:

- **s**: The synch indicator is initially `?` before the clock is set, but turns to space when all nine digits of the timecode are correctly set and the decoder is synchronized to the station within 125 µs.
- **q**: The quality character is a four-bit hexadecimal code showing which alarms have been raised. Each bit is associated with a specific alarm condition according to the following:
  - **0x8**: Synch alarm. The decoder is not synchronized to the station within 125 µs.
  - **0x4**: Digit error alarm. Less than nine decimal digits were found in the last minute.
  - **0x2**: Error alarm. More than 40 data bit errors were found in the last minute.
  - **0x1**: Compare alarm. A maximum-likelihood digit failed to agree with the current associated clock digit in the last minute.

It is important to note that one or more of the above alarms does not necessarily indicate a clock error, but only that the decoder has detected a marginal condition.

- **yyyy ddd hh:mm:ss**: The timecode format itself is self explanatory. Since the driver latches the on-time epoch directly from the second synch pulse, the seconds fraction is always zero. Although the transmitted timecode includes only the year of century, the Gregorian year is augmented by 2000.
- **l**: The leap second warning is normally space, but changes to `L` if a leap second is to occur at the end of the month.
- **d**: The DST state is `S` or `D` when standard time or daylight time is in effect, respectively. The state is `I` or `O` when daylight time is about to go into effect or out of effect, respectively.
- **du**: The DUT sign and magnitude shows the current UT1 offset relative to the displayed UTC time, in deciseconds.
- **lset**: Before the clock is set, the interval since last set is the number of minutes since the driver was started; after the clock is set, this is number of minutes since the decoder was last synchronized to the station within 125 µs.
- **agc**: The audio gain shows the current codec gain setting in the range 0 to 255. Ordinarily, the receiver audio gain control should be set for a value midway in this range.
- **ident**: The station identifier shows the station, `WVf` for WWV or `WHf` for WWVH, and frequency ‘f’ being tracked. If neither station is heard on any frequency, the reference identifier shows `NONE`.
- **metric**: The signal metric described above from 0 (no signal) to 100 (best).
- **errs**: The bit error counter is useful to determine the quality of the data signal received in the most recent minute. It is normal to drop a couple of data bits even under good signal conditions and increasing numbers as conditions worsen. While the decoder performs moderately well even with half the bits are in error in any minute, usually by that point the metric drops below threshold and the decoder switches to a different frequency.
- **freq**: The frequency offset is the current estimate of the codec frequency offset to within 0.1 PPM. This may wander a bit over the day due to local temperature fluctuations and propagation conditions.
- **avg**: The averaging time is the interval between frequency updates in powers of two to a maximum of 1024 s. Attainment of the maximum indicates the driver is operating at the best possible resolution in time and frequency.

An example timecode is:

```
0 2000 006 22:36:00 S +3 1 115 WV20 86 5 66.4 1024
```
Here the clock has been set and no alarms are raised. The year, day and time are displayed along with no leap warning, standard time and DUT +0.3 s. The clock was set on the last minute, the AGC is safely in the middle of the range 0-255, and the receiver is tracking WWV on 20 MHz. Good receiving conditions prevail, as indicated by the metric 86 and 5 bit errors during the last minute. The current frequency is 66.4 PPM and the averaging interval is 1024 s, indicating the maximum precision available.

### 7.4.13 Fudge Factors

- **time1 time** Specifies the propagation delay for WWV (40:40:49.0N 105:02:27.0W), in seconds and fraction, with default 0.0.

- **time2 time** Specifies the propagation delay for WWVH (21:59:26.0N 159:46:00.0W), in seconds and fraction, with default 0.0.

- **stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

- **refid string** Ordinarily, this field specifies the driver reference identifier; however, the driver sets the reference identifier automatically as described above.

- **flag1 0 | 1** Not used by this driver.

- **flag2 0 | 1** Specifies the microphone port if set to zero or the line-in port if set to one. It does not seem useful to specify the compact disc player port.

- **flag3 0 | 1** Enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.

- **flag4 0 | 1** Enable verbose clockstats recording if set.
8.1 Configuration File Definition (Advanced)

8.1.1 Synopsis

The NTP configuration process is driven by a phrase-structure grammar which is used to specify the format of the configuration commands and the actions needed to build an abstract syntax tree (AST). The grammar is fed to a parser generator (Bison) which produces a parser for the configuration file.

The generated parser is used to parse an NTP configuration file and check it for syntax and semantic errors. The result of the parse is an AST, which contains a representation of the various commands and options. This AST is then traversed to set up the NTP daemon to the correct configuration.

This document is intended for developers who wish to modify the configuration code and/or add configuration commands and options. It contains a description of the files used in the configuration process as well as guidelines on how to construct them.

8.1.2 Files

A brief description of the files used by the configuration code is given below:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntp_config.y</td>
<td>This file is a Bison source file that contains the phrase-structure grammar and the actions that need to be performed to generate an AST.</td>
</tr>
<tr>
<td>ntp_config.c</td>
<td>This file contains the major chunk of the configuration code. It contains all the functions that are called for building the AST as well as the functions that are needed for traversing the AST.</td>
</tr>
<tr>
<td>ntp_config.h</td>
<td>This file is the header file for ntp_config.c. It mainly contains the structure definitions needed to build the AST.</td>
</tr>
<tr>
<td>ntp_scanner.c</td>
<td>This file contains the code for a simple lexical analyzer. This file is directly included into the ntp_config.c file since this code is only used by the configuration code. The most important function in this file is yylex, which is called by the generated parser to get the next token on the input line.</td>
</tr>
<tr>
<td>ntp_data_structures.c</td>
<td>This file contains a generic implementation of a priority queue and a simple queue. This code can be used to create a queue for any structure.</td>
</tr>
<tr>
<td>ntp_data_structures.h</td>
<td>Header file contains the structure declarations and function prototypes needed to use the data structures defined in ntp_data_structures.c. This file forms the public interface of the data structures.</td>
</tr>
<tr>
<td>ntp_config.tab.c</td>
<td>This file is generated by Bison from the ntp_config.y file. This file is also included directly into the configuration code.</td>
</tr>
</tbody>
</table>
8.1.3 High-Level Description

A high-level description of the configuration process showing where all the files fit in is given below:

The scanner reads in an NTP configuration file and converts it into tokens. The Bison generated parser reads these tokens and converts them into an AST. The AST traverser consists of a set of functions that configure parts of NTP on the basis of what is on the tree. A more detailed description of these parts and the files used is given below:

8.1.4 Detailed Description

ntp_scanner.c This file contains the scanner. The scanner is a small program that converts an input NTP configuration file into a set of tokens that correspond to lexemes in the input. Lexemes are strings in the input, delimited by whitespace and/or special characters. Tokens are basically unique integers that represent these lexemes. A different token is generated for each reserved word and special character in the input. There are two main functions in the public interface of this file:

int yylex() This function is called yylex for historical reasons; lex is a program that takes a set of regular expressions and generates a scanner that returns tokens corresponding to those regular expressions. The name of the generated function is called yylex. We aren’t using lex because it requires linking against an external library and we didn’t want to increase the compile-time requirements of NTP. History lessons aside, this function basically checks to see if the next input character is a special character as defined in the array char special_char[]. (The function int is_special(char ch), can be used for this.) If yes, the special character is returned as the token. If not, a set of characters is read until the next whitespace or special character is encountered. This set of characters forms the lexeme; yylex then checks whether this lexeme is an integer, a double, an IP address or a reserved word. If yes, the corresponding token is returned. If not, a token for a string is returned as the default token.

struct state *create_keyword_scanner(struct key_tok *keyword_list) This function takes a list of (keyword, token) pairs and converts them into a trie that can recognize the keywords (reserved words). Every time the scanner reads a lexeme, it compares it against the list of reserved words. If it finds a match, it returns the corresponding token for that keyword.

ntp_data_structures.c This file contains an implementation of a generic priority queue and FIFO queue. By generic, we mean that these queues can hold elements of any type (integers, user-defined structs, etc.), provided that these elements are allocated on the heap using the function void *get_node(size_t size). Note that the prototype for this function is exactly the same as that of malloc and that it can be used in the exact same way. Behind the scenes, get_node calls malloc to allocate size plus some extra memory needed for bookkeeping. The allocated memory can be freed using the function void free_node (void *my_node). In addition to these two functions, the public interface of this file contains the following functions:

queue *create_priority_queue(int (*get_order)(void *, void*)) This function creates a priority queue in which the order of the elements is determined by the get_order function that is passed
as input to the priority queue. The `get_order` function should return positive if the priority of the first element is less than the priority of the second element.

```c
queue *create_queue(void) This function creates a FIFO queue. It basically calls the create_priority_queue function with the get_fifo_order function as its argument.
```

```c
void destroy_queue(queue *my_queue) This function deletes my_queue and frees up all the memory allocated to it an its elements.
```

```c
int empty(queue *my_queue) This function checks to see if my_queue is empty. Returns true if my_queue does not have any elements, else it returns false.
```

```c
queue *enqueue(queue *my_queue, void *my_node) This function adds an element, my_node, to a queue, my_queue. my_node must be allocated on the heap using the get_node function instead of malloc.
```

```c
void *dequeue(queue *my_queue) This function returns the element at the front of the queue. This element will be element with the highest priority.
```

```c
int get_no_of_elements(queue *my_queue) This function returns the number of elements in my_queue.
```

```c
void append_queue(queue *q1, queue *q2) This function adds all the elements of q2 to q1. The queue q2 is destroyed in the process.
```

**ntp_config.y** This file is structured as a standard Bison file and consists of three main parts, separated by `%%`:

1. The prologue and bison declarations: This section contains a list of the terminal symbols, the non-terminal symbols and the types of these symbols.
2. The rules section: This section contains a description of the actual phrase-structure rules that are used to parse the configuration commands. Each rule consists of a left-hand side (LHS), a right-hand side (RHS) and an optional action. As is standard with phrase-structure grammars, the LHS consists of a single non-terminal symbol. The RHS can contain both terminal and non-terminal symbols, while the optional action can consist of any arbitrary C code.
3. The epilogue: This section is left empty on purpose. It is traditionally used to code the support functions needed to build the ASTs. Since, we have moved all the support functions to `ntp_config.c`, this section is left empty.

### 8.1.5 Prologue and Bison Declarations

All the terminal symbols (also known as tokens) have to be declared in the prologue section. Note that terminals and non-terminals may have values associated with them and these values have types. (More on this later). An unnamed union has to be declared with all the possible types at the start of the prologue section. For example, we declare the following union at the start of the `ntp_config.y` file:

```c
%union {
    char *String;
    double Double;
    int Integer;
    void *VoidPtr;
    queue *Queue;
    struct attr_val *Attr_val;
    struct address_node *Address_node;
    struct setvar_node *Set_var;
    /* Simulation types */
    server_info *Sim_server;
    script_info *Sim_script;
}
```
Some tokens may not have any types. For example, tokens that correspond to reserved words do not usually have types as they simply indicate that a reserved word has been read in the input file. Such tokens have to be declared as follows:

```
%token T_Discard
%token T_Dispersion
```

Other tokens do have types. For example, a `T_Double` token is returned by the scanner whenever it sees a floating-point double in the configuration file. The value associated with the token is the actual number that was read in the configuration file and its type (after conversion) is double. Hence, the token `T_Double` will have to be declared as follows in the prologue of `ntp_config.y` file:

```
%token <Double> T_Double
```

Note that the declaration given in the angled brackets is not `double` but `Double`, which is the name of the variable given in the `%union {}` declaration above.

Finally, non-terminal symbols may also have values associated with them, which have types. This is because Bison allows non-terminal symbols to have actions associated with them. Actions may be thought of as small functions which get executed whenever the RHS of a non-terminal is detected. The return values of these functions are the values associated with the non-terminals. The types of the non-terminals are specified with a `%type` declaration as shown below:

```
%type <Queue> address_list
%type <Integer> boolean
```

The `%type` declaration may be omitted for non-terminals that do not return any value and do not have type information associated with them.

### 8.1.6 The Rules Section

The rule section only consists of phrase-structure grammar rules. Each rule typically has the following format:

```
LHS : RHS [{ Actions }] ;
```

where LHS consists of a single non-terminal symbol and the RHS consists of one or more terminal and non-terminal symbols. The `Actions` are optional and may consist of any number of arbitrary C statements. Note that Bison can only process LALR(1) grammars, which imposes additional restrictions on the kind of rules that can be specified. Examples of rules are shown below:

```
orphan_mode_command
  : T_Tos tos_option_list
    { append_queue(my_config.orphan_cmds, $2); }
    ;

  tos_option_list
    : tos_option_list tos_option { $$ = enqueue($1, $2); }
    | tos_option { $$ = enqueue_in_new_queue($1); }
    ;
```

The `$n` notation, where `n` is an integer, is used to refer to the value of a terminal or non-terminal symbol. All terminals and non-terminal symbols within a particular rule are numbered (starting from 1) according to the order in which they appear within the RHS of a rule. `$$` is used to refer to the value of the LHS terminal symbol - it is used to return a value for the non-terminal symbol specified in the LHS of the rule.
8.1.7 Invoking Bison

Bison needs to be invoked in order to convert the ntp_config.y file into a C source file. To invoke Bison, simply enter the command:

```
bison ntp_config.y
```

at the command prompt. If no errors are detected, an ntp_config.tab.c file will be generated by default. This generated file can be directly included into the ntp_config.c file.

If Bison report shift-reduce errors or reduce-reduce errors, it means that the grammar specified using the rules in not LALR(1). To debug such a grammar, invoke Bison with a -v switch, as shown below. This will generate a ntp_config.output file, which will contain a description of the generated state machine, together with a list of states that have shift-reduce/reduce-reduce conflicts. You can then change the rules to remove such conflicts.

```
bison -v ntp_config.y
```

For more information, refer to the Bison manual.

ntp_config.c

This file contains the major chunk of the configuration code including all the support functions needed for building and traversing the ASTs. As such, most of the functions in this file can be divided into two groups:

1. Functions that have a create_ prefix. These functions are used to build a node of the AST.
2. Functions that have a config_ prefix. These functions are used to traverse the AST and configure NTP according to the nodes present on the tree.

8.1.8 Guidelines for Adding Configuration Commands

The following steps may be used to add a new configuration command to the NTP reference implementation:

1. Write phrase-structure grammar rules for the syntax of the new command. Add these rules to the rules section of the ntp_config.y file.
2. Write the action to be performed on recognizing the rules. These actions will be used to build the AST.
3. If new reserved words are needed, add these to the struct key_tok keyword_list[] structure in the ntp_config.c file. This will allow the scanner to recognize these reserved words and generate the desired tokens on recognizing them.
4. Specify the types of all the terminals and non-terminal symbols in the prologue section of the ntp_config.c file.
5. Write a function with a config_ prefix that will be executed for this new command. Make sure this function is called in the config_ntpd() function.

8.2 Event Messages and Status Words

This page lists the status words, event messages and error codes used for ntpd reporting and monitoring. Status words are used to display the current status of the running program. There is one system status word and a peer status word for each association. There is a clock status word for each association that supports a reference clock. There is a flash code for each association which shows errors found in the last packet received (pkt) and during protocol processing (peer). These are commonly viewed using the ntpq program.

Significant changes in program state are reported as events. There is one set of system events and a set of peer events for each association. In addition, there is a set of clock events for each association that supports a reference clock.
Events are normally reported to the protostats monitoring file and optionally to the system log. In addition, if the trap facility is configured, events can be reported to a remote program that can page an administrator.

This page also includes a description of the error messages produced by the Autokey protocol. These messages are normally sent to the cryptostats monitoring file.

In the following tables the Event Field is the status or event code assigned and the Message Field a short string used for display and event reporting. The Description field contains a longer explanation of the status or event. Some messages include additional information useful for error diagnosis and performance assessment.

### 8.2.1 System Status Word

The system status word consists of four fields LI (0-1), Source (2-7), Count (8-11) and Event (12-15). It is reported in the first line of the *rv* display produced by the *ntpq* program.

<table>
<thead>
<tr>
<th>Leap</th>
<th>Source</th>
<th>Count</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Leap Field displays the system leap indicator bits coded as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>leap_none</td>
<td>normal synchronized state</td>
</tr>
<tr>
<td>1</td>
<td>leap_add_sec</td>
<td>insert second after 23:59:59 of the current day</td>
</tr>
<tr>
<td>2</td>
<td>leap_del_sec</td>
<td>delete second 23:59:59 of the current day</td>
</tr>
<tr>
<td>3</td>
<td>leap_alarm</td>
<td>never synchronized</td>
</tr>
</tbody>
</table>

The Source Field displays the current synchronization source coded as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sync_unspec</td>
<td>not yet synchronized</td>
</tr>
<tr>
<td>1</td>
<td>sync_pps</td>
<td>pulse-per-second signal (Cs, Ru, GPS, etc.)</td>
</tr>
<tr>
<td>2</td>
<td>sync_lf_radio</td>
<td>VLF/LF radio (WWVB, DCF77, etc.)</td>
</tr>
<tr>
<td>3</td>
<td>sync_hf_radio</td>
<td>MF/HF radio (WWV, etc.)</td>
</tr>
<tr>
<td>4</td>
<td>sync_uhf_radio</td>
<td>VHF/UHF radio/satellite (GPS, Galileo, etc.)</td>
</tr>
<tr>
<td>5</td>
<td>sync_local</td>
<td>local timecode (IRIG, LOCAL driver, etc.)</td>
</tr>
<tr>
<td>6</td>
<td>sync_ntp</td>
<td>NTP</td>
</tr>
<tr>
<td>7</td>
<td>sync_other</td>
<td>other (IEEE 1588, openntp, crony, etc.)</td>
</tr>
<tr>
<td>8</td>
<td>sync_wristwatch</td>
<td>eyeball and wristwatch</td>
</tr>
<tr>
<td>9</td>
<td>sync_telephone</td>
<td>telephone modem (ACTS, PTB, etc.)</td>
</tr>
</tbody>
</table>

The Count Field displays the number of events since the last time the code changed. Upon reaching 15, subsequent events with the same code are ignored.

The Event Field displays the most recent event message coded as follows:
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>unspecified</td>
<td>unspecified</td>
</tr>
<tr>
<td>01</td>
<td>freq_not_set</td>
<td>frequency file not available</td>
</tr>
<tr>
<td>02</td>
<td>freq_set</td>
<td>frequency set from frequency file</td>
</tr>
<tr>
<td>03</td>
<td>spike_detect</td>
<td>spike detected</td>
</tr>
<tr>
<td>04</td>
<td>freq_mode</td>
<td>initial frequency training mode</td>
</tr>
<tr>
<td>05</td>
<td>clock_sync</td>
<td>clock synchronized</td>
</tr>
<tr>
<td>06</td>
<td>restart</td>
<td>program restart</td>
</tr>
<tr>
<td>07</td>
<td>panic_stop</td>
<td>clock error more than 600 s</td>
</tr>
<tr>
<td>08</td>
<td>no_system_peer</td>
<td>no system peer</td>
</tr>
<tr>
<td>09</td>
<td>leap_armed</td>
<td>leap second armed from file or Autokey</td>
</tr>
<tr>
<td>0a</td>
<td>leap_disarmed</td>
<td>leap second disarmed</td>
</tr>
<tr>
<td>0b</td>
<td>leap_event</td>
<td>leap event</td>
</tr>
<tr>
<td>0c</td>
<td>clock_step</td>
<td>clock stepped</td>
</tr>
<tr>
<td>0d</td>
<td>kern</td>
<td>kernel information message</td>
</tr>
<tr>
<td>0e</td>
<td>TAI...</td>
<td>leapsecond values update from file</td>
</tr>
<tr>
<td>0f</td>
<td>stale leapsecond values</td>
<td>new NIST leapseconds file needed</td>
</tr>
</tbody>
</table>

### 8.2.2 Peer Status Word

The peer status word consists of four fields: Status (0-4), Select (5-7), Count (8-11) and Code (12-15). It is reported in the first line of the `rv associd` display produced by the `ntpq` program.

<table>
<thead>
<tr>
<th>Status</th>
<th>Select</th>
<th>Count</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Status Field displays the peer status code bits in hexadecimal; each bit is an independent flag. (Note this field is 5 bits wide, and combines with the the 3-bit-wide Select Field to create the first full byte of the peer status word.) The meaning of each bit in the Status Field is listed in the following table:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>bcst</td>
<td>broadcast association</td>
</tr>
<tr>
<td>10</td>
<td>reach</td>
<td>host reachable</td>
</tr>
<tr>
<td>20</td>
<td>auth</td>
<td>authentication ok</td>
</tr>
<tr>
<td>40</td>
<td>authenb</td>
<td>authentication enabled</td>
</tr>
<tr>
<td>80</td>
<td>config</td>
<td>persistent association</td>
</tr>
</tbody>
</table>

The Select Field displays the current selection status. (The T Field in the following table gives the corresponding tally codes used in the `ntpq peers` display.) The values are coded as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>T</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sel_reject</td>
<td></td>
<td>discarded as not valid (TEST10-TEST13)</td>
</tr>
<tr>
<td>1</td>
<td>sel_falsetick</td>
<td>x</td>
<td>discarded by intersection algorithm</td>
</tr>
<tr>
<td>2</td>
<td>sel_excess</td>
<td>.</td>
<td>discarded by table overflow (not used)</td>
</tr>
<tr>
<td>3</td>
<td>sel_outlier</td>
<td>-</td>
<td>discarded by the cluster algorithm</td>
</tr>
<tr>
<td>4</td>
<td>sel_candidate</td>
<td>+</td>
<td>included by the combine algorithm</td>
</tr>
<tr>
<td>5</td>
<td>sel_backup</td>
<td>#</td>
<td>backup (more than tos maxclock sources)</td>
</tr>
<tr>
<td>6</td>
<td>sel_sys.peer</td>
<td>*</td>
<td>system peer</td>
</tr>
<tr>
<td>7</td>
<td>sel_pps.peer</td>
<td>o</td>
<td>PPS peer (when the prefer peer is valid)</td>
</tr>
</tbody>
</table>

The Count Field displays the number of events since the last time the code changed. Upon reaching 15, subsequent events with the same code are ignored.

The Event Field displays the most recent event message coded as follows:

### 8.2. Event Messages and Status Words
Code | Message | Description
--- | --- | ---
01 | mobilize | association mobilized
02 | demobilize | association demobilized
03 | unreachable | server unreachable
04 | reachable | server reachable
05 | restart | association restart
06 | no_reply | no server found (ntpd mode)
07 | rate_exceeded | rate exceeded (kiss code RATE)
08 | access_denied | access denied (kiss code DENY)
09 | leap_armed | leap armed from server LI code
0a | sys_peer | become system peer
0b | clock_event | see clock status word
0c | bad_auth | authentication failure
0d | popcorn | popcorn spike suppressor
0e | interleave_mode | entering interleave mode
0f | interleave_error | interleave error (recovered)

### 8.2.3 Clock Status Word

The clock status word consists of four fields: Unused (0-7), Count (8-11) and Code (12-15). It is reported in the first line of the `clockvar` `associd` display produced by the `ntpq` program.

<table>
<thead>
<tr>
<th>Unused</th>
<th>Count</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Count Field displays the number of events since the last `lockvar` command, while the Event Field displays the most recent event message coded as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>clk_unspe</td>
<td>nominal</td>
</tr>
<tr>
<td>01</td>
<td>clk_noreply</td>
<td>no reply to poll</td>
</tr>
<tr>
<td>02</td>
<td>clk_badformat</td>
<td>bad timecode format</td>
</tr>
<tr>
<td>03</td>
<td>clk_fault</td>
<td>hardware or software fault</td>
</tr>
<tr>
<td>04</td>
<td>clk_bad_signal</td>
<td>signal loss</td>
</tr>
<tr>
<td>05</td>
<td>clk_bad_date</td>
<td>bad date format</td>
</tr>
<tr>
<td>06</td>
<td>clk_bad_time</td>
<td>bad time format</td>
</tr>
</tbody>
</table>

When the clock driver sets the code to a new value, a `clock_alarm` (11) peer event is reported.

### 8.2.4 Flash Status Word

The flash status word is displayed by the `ntpq` program `rv` command. It consists of a number of bits coded in hexadecimal as follows:
### 8.2.5 Kiss Codes

Kiss codes are used in kiss-o'-death (KoD) packets, billboard displays and log messages. They consist of a string of four zero-padded ASCII characters. In practice they are informal and tend to change with time and implementation. Some of these codes can appear in the reference identifier field in `ntpq` billboards. Following is the current list:

<table>
<thead>
<tr>
<th>Code</th>
<th>Tag</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>TEST1</td>
<td>pkt_dup</td>
<td>duplicate packet</td>
</tr>
<tr>
<td>0002</td>
<td>TEST2</td>
<td>pkt_bogus</td>
<td>bogus packet</td>
</tr>
<tr>
<td>0004</td>
<td>TEST3</td>
<td>pkt_unsync</td>
<td>server not synchronized</td>
</tr>
<tr>
<td>0008</td>
<td>TEST4</td>
<td>pkt_denied</td>
<td>access denied</td>
</tr>
<tr>
<td>0010</td>
<td>TEST5</td>
<td>pkt_auth</td>
<td>authentication failure</td>
</tr>
<tr>
<td>0020</td>
<td>TEST6</td>
<td>pkt_stratum</td>
<td>invalid leap or stratum</td>
</tr>
<tr>
<td>0040</td>
<td>TEST7</td>
<td>pkt_header</td>
<td>header distance exceeded</td>
</tr>
<tr>
<td>0080</td>
<td>TEST8</td>
<td>pkt_autokey</td>
<td>Autokey sequence error</td>
</tr>
<tr>
<td>0100</td>
<td>TEST9</td>
<td>pkt_crypto</td>
<td>Autokey protocol error</td>
</tr>
<tr>
<td>0200</td>
<td>TEST10</td>
<td>peer_stratum</td>
<td>invalid header or stratum</td>
</tr>
<tr>
<td>0400</td>
<td>TEST11</td>
<td>peer_dist</td>
<td>distance threshold exceeded</td>
</tr>
<tr>
<td>0800</td>
<td>TEST12</td>
<td>peer_loop</td>
<td>synchronization loop</td>
</tr>
<tr>
<td>1000</td>
<td>TEST13</td>
<td>peer_unreach</td>
<td>unreachable or nonselect</td>
</tr>
</tbody>
</table>

### 8.2.6 Crypto Messages

These messages are sent to the `cryptostats` file when an error is detected in the Autokey protocol.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>bad_format</td>
<td>bad extension field format or length</td>
</tr>
<tr>
<td>02</td>
<td>bad_timestamp</td>
<td>bad timestamp</td>
</tr>
<tr>
<td>03</td>
<td>bad_filestamp</td>
<td>bad filestamp</td>
</tr>
<tr>
<td>04</td>
<td>bad_public_key</td>
<td>bad or missing public key</td>
</tr>
<tr>
<td>05</td>
<td>bad_digest</td>
<td>unsupported digest type</td>
</tr>
<tr>
<td>06</td>
<td>bad_identity</td>
<td>unsupported identity type</td>
</tr>
<tr>
<td>07</td>
<td>bad_signlength</td>
<td>bad signature length</td>
</tr>
<tr>
<td>08</td>
<td>bad_signature</td>
<td>extension field signature not verified</td>
</tr>
<tr>
<td>09</td>
<td>cert_not_verified</td>
<td>certificate signature not verified</td>
</tr>
<tr>
<td>0a</td>
<td>cert_expired</td>
<td>host certificate expired</td>
</tr>
<tr>
<td>0b</td>
<td>bad_cookie</td>
<td>bad or missing cookie</td>
</tr>
<tr>
<td>0c</td>
<td>bad_leapseconds</td>
<td>bad or missing leapseconds values</td>
</tr>
<tr>
<td>0d</td>
<td>cert_missing</td>
<td>bad or missing certificate</td>
</tr>
<tr>
<td>0e</td>
<td>bad_group_key</td>
<td>bad or missing group key</td>
</tr>
<tr>
<td>0f</td>
<td>proto_error</td>
<td>protocol error</td>
</tr>
</tbody>
</table>

### 8.3 Kernel Model for Precision Timekeeping

The technical report \(^1\), which is a revision and update of an earlier report \(^2\), describes an engineering model for a precision clock discipline function for a generic operating system. The model is the same hybrid phase/frequency-lock feedback loop used by `ntpd`, but implemented in the kernel. The code described in \(^2\) is included in Solaris and Digital/Compaq/HP Tru64. It provides two system calls `ntp_gettime()` and `ntp_adjtime()` and can discipline the system clock with microsecond resolution. However, newer hardware and kernels with the same system calls can discipline the clock with nanosecond resolution. The new code described in \(^3\) is in FreeBSD, Linux and Tru64. The software and documentation, including a simulator used to verify correct behavior, but not involving licensed code, is available in the nanokernel.tar.gz distribution.

Ordinarily, the kernel clock discipline function is used with the NTP daemon, but could be used for other purposes. The `ntptime` utility program can be used to control it manually.

The kernel model also provides support for an external precision timing source, such as described in the Pulse-per-second (PPS) Signal Interfacing page. The new system calls are used by the PPSAPI interface and in turn by the PPS Clock Discipline driver (type 22) to provide synchronization limited in principle only by the accuracy and stability of the external timing source. Typical results with the PPS signal from a GPS receiver and a modern computer are in the 3 \(\mu\)s range.

#### References

### 8.4 `ntpd` System Log Messages

You have come here because you found a cryptic message in the system log. This page by no means lists all messages that might be found, since new ones come and old ones go. Generally, however, the most common ones will be found here. They are listed by program module and log severity code in bold: LOG_ERR, LOG_NOTICE and LOG_INFO.


Most of the time \texttt{LOG\_ERR} messages are fatal, but often \texttt{ntpd} limps onward in the hopes of discovering more errors. The \texttt{LOG\_NOTICE} messages usually mean the time has changed or some other condition that probably should be noticed. The \texttt{LOG\_INFO} messages usually say something about the system operations, but do not affect the time.

In the following a ‘?’ character stands for text in the message. The meaning should be clear from context.

### 8.4.1 Protocol Module

**\texttt{LOG\_ERR}**

\texttt{buffer overflow ?} Fatal error. An input packet is too long for processing.

**\texttt{LOG\_NOTICE}**

\texttt{no reply; clock not set} In \texttt{ntpd} mode no servers have been found. The server(s) and/or network may be down. Standard debugging procedures apply.

**\texttt{LOG\_INFO}**

\texttt{proto\_config: illegal item ?, value ?} Program error. Bugs can be reported here.

\texttt{receive: autokey requires two-way communication} Configuration error on the \texttt{broadcastclient} command.

\texttt{receive: server server maximum rate exceeded} A kiss-o’death packet has been received. The transmit rate is automatically reduced.

\texttt{pps sync enabled} The PPS signal has been detected and enabled.

\texttt{transmit: encryption key ? not found} The encryption key is not defined or not trusted.

\texttt{precision = ? usec} This reports the precision measured for this machine.

\texttt{using 10ms tick adjustments} Gotcha for some machines with dirty rotten clock hardware.

\texttt{no servers reachable} The system clock is running on internal batteries. The server(s) and/or network may be down.

### 8.4.2 Clock Discipline Module

**\texttt{LOG\_ERR}**

\texttt{time correction of ? seconds exceeds sanity limit (?); set clock manually to the correct UTC time.} Fatal error. Better do what it says, then restart the daemon. Be advised NTP and Unix know nothing about local time zones. The clock must be set to Coordinated Universal Time (UTC). Believe it; by international agreement abbreviations are in French and descriptions are in English.

\texttt{sigaction() fails to save SIGSYS trap: ?}

\texttt{sigaction() fails to restore SIGSYS trap: ?}

Program error. Bugs can be reported here.

### 8.4. \texttt{ntpd} System Log Messages
LOG_NOTICE

*frequency error ? exceeds tolerance 500 PPM* The hardware clock frequency error exceeds the rate the kernel can correct. This could be a hardware or a kernel problem.

*time slew ? s* The time error exceeds the step threshold and is being slewed to the correct time. You may have to wait a very long time.

*time reset ? s* The time error exceeds the step threshold and has been reset to the correct time. Computer scientists don’t like this, but they can set the `ntpd -x` option and wait forever.

**kernel time sync disabled ?** The kernel reports an error. See the codes in the `timex.h` file.

**pps sync disabled** The PPS signal has died, probably due to a dead radio, broken wire or loose connector.

LOG_INFO

**kernel time sync status ?** For information only. See the codes in the `timex.h` file.

8.4.3 Cryptographic Module

LOG_ERR

```plaintext
%cert_parse
%cert_sign
%crypto_cert
%crypto_encrypt
%crypto_gq
%crypto_iff
%crypto_key
%crypto_mv
%crypto_setup
%make_keys
```

Usually fatal errors. These messages display error codes returned from the OpenSSL library. See the OpenSSL documentation for explanation.

```plaintext
%crypto_setup: certificate ? is trusted, but not self signed
%crypto_setup: certificate ? not for this host
%crypto_setup: certificate file ? not found or corrupt
%crypto_setup: host key file ? not found or corrupt
%crypto_setup: host key is not RSA key type
%crypto_setup: random seed file ? not found
%crypto_setup: random seed file not specified
```

Fatal errors. These messages show problems during the initialization procedure.
LOG_INFO

cert_parse: expired ?
cert_parse: invalid issuer ?
cert_parse: invalid signature ?
cert_parse: invalid subject ?

There is a problem with a certificate. Operation cannot proceed until the problem is fixed. If the certificate is local, it can be regenerated using the ntp-keygen program. If it is held somewhere else, it must be fixed by the holder.

crypto_?: defective key
crypto_?: invalid filestamp
crypto_?: missing challenge
crypto_?: scheme unavailable

There is a problem with the identity scheme. Operation cannot proceed until the problem is fixed. Usually errors are due to misconfiguration or an orphan association. If the latter, ntpd will usually time out and recover by itself.

crypto_cert: wrong PEM type ? The certificate does not have MIME type CERTIFICATE. You are probably using the wrong type from OpenSSL or an external certificate authority.
crypto_ident: no compatible identity scheme found Configuration error. The server and client identity schemes are incompatible.
crypto_tai: kernel TAI update failed The kernel does not support this function. You may need a new kernel or patch.

8.5 PPSAPI Interface for Precision Time Signals

RFC 2783 describes the PPSAPI application programming interface for external precision time signals, such as the pulse-per-second (PPS) signal generated by some radio clocks and cesium oscillators. The PPSAPI provides a generic capability in the ubiquitous Unix kernel which can be used for a wide variety of measurement applications, including network time synchronization and related experiments. The hardware to do this requires only a serial port and a modem control lead, such as the data carrier detect (DCD) lead, which can be driven by an external source via a level converter/pulse generator such as described on the Pulse-per-second (PPS) Signal Interfacing page. In some systems a parallel port can be used for the same purpose.

The PPSAPI interface defined in RFC 2783 is the only PPS interface supported in NTP Version 4. The PPSAPI is supported in stock FreeBSD and, with the addition of the PPSkit kernel module, in Linux.

The special header file /usr/include/sys/timepps.h implements the PPSAPI using whatever primitives are available in each architecture and operating system. It obsoletes previous APIs based on the tty_clock and ppsclock line disciplines and streams modules, which are no longer supported.

The PPS Clock Discipline driver (type 22) uses the PPSAPI in conjunction with a local radio clock or remote NTP server as a reference clock. The driver can also use the PPSAPI as an interface directly to the kernel PPS facility as described on the Kernel Model for Precision Timekeeping page.
8.5.1 PPSAPI Application Program Interface

The PPSAPI interface provides the following functions:

- **time_pps_create**
  - Creates a PPS interface instance and returns a handle to it.

- **time_pps_destroy**
  - Destroys a PPS interface and returns the resources used.

- **time_pps_setparams**
  - Sets the parameters associated with a PPS interface instance, including offsets to be automatically added to captured timestamps.

- **time_pps_getparams**
  - Returns the parameters associated with a PPS interface instance.

- **time_pps_getcap**
  - Returns the capabilities of the current interface and kernel implementation.

- **time_pps_fetch**
  - Returns the current timestamps associated with a PPS interface instance in either nanoseconds and nanoseconds (Unix `timespec`) or seconds and fraction (NTP) format.

- **time_pps_kcbind**
  - If kernel PPS processing is supported, this binds the support to the associated PPS interface instance.

The entire PPS interface functionality is currently provided by inline code in the `timepps.h` header file. While not all implementations support the full PPSAPI specification, they do support all the functions required for the PPS driver described next. The FreeBSD, Linux and Solaris implementations can be used with the stock kernels provided with those systems; however, the Tru64 and SunOS kernels require additional functions not provided in the stock kernels. Solaris users are cautioned that these functions operate improperly in Solaris versions prior to 2.8 with patch Generic_108528-02. Header files for other systems can be found via the web at nanokernel.tar.gz.

8.6 Pulse-Per-Second (PPS) Signal Interfacing

Most radio clocks are connected using a serial port operating at speeds of 9600 bps. The accuracy using typical timecode formats, where the on-time epoch is indicated by a designated ASCII character such as carriage-return `<cr>`, is normally limited to 100 μs. Using carefully crafted averaging techniques, the NTP algorithms can whittle this down to a few tens of microseconds. However, some radios produce a pulse-per-second (PPS) signal which can be used to improve the accuracy to a few microseconds. This page describes the hardware and software necessary for NTP to use the PPS signal.

The PPS signal can be connected in either of two ways. On FreeBSD systems (with the PPS_SYNC and pps kernel options) it can be connected directly to the ACK pin of a parallel port. This is the preferred way, as it requires no additional hardware. Alternatively, it can be connected via the DCD pin of a serial port. However, the PPS signal levels are usually incompatible with the serial port interface signals. Note that NTP no longer supports connection via the RD pin of a serial port.
8.6.1 Gadget Box

The gadget box shown above is assembled in a 5”x3”x2” aluminum minibox containing the circuitry, serial connector and optional 12-V power connector. A complete set of schematics, PCB artwork, drill templates can be obtained via the web from ftp.udel.edu as gadget.tar.Z.

The gadget box includes two subcircuits. One of these converts a TTL positive edge into a fixed-width pulse at EIA levels and is for use with a timecode receiver or precision oscillator with a TTL PPS output. The other converts the timecode modulation broadcast by Canadian time/frequency standard station CHU into a 300-bps serial character stream at EIA levels and is for use with the Radio CHU Audio Demodulator/Decoder driver.

8.6.2 Operating System Support

Both the serial and parallel port connection require operating system support, which is available in a few operating systems, including FreeBSD, Linux (with PPSkit patch) and Solaris. Support on an experimental basis is available for several other systems, including SunOS and HP/Compaq/Digital Tru64. The kernel interface described on the PPSAPI Interface for Precision Time Signals page is the only interface currently supported. Older PPS interfaces based on the ppsclock and tty_clk streams modules are no longer supported. The interface consists of the timepps.h header file which is specific to each system. It is included automatically when the distribution is built.

8.6.3 PPS Driver

PPS support requires is built into some drivers, in particular the WWVB and NMEA drivers, and may be added to other drivers in future. Alternatively, the PPS driver described on the Type 22 PPS Clock Discipline page can be used. It operates in conjunction with another source that provides seconds numbering. The selected source is designate a prefer peer, as using the prefer option, as described on the Mitigation Rules and the prefer Keyword page. The prefer peer is ordinarily the radio clock that provides the PPS signal, but in principle another radio clock or even a remote Internet server could be designated preferred Note that the pps configuration command has been obsoleted by this driver.
8.6.4 Using the Pulse-per-Second (PPS) Signal

The PPS signal can be used in either of two ways, one using the NTP grooming and mitigation algorithms and the other using the kernel PPS signal support described in the Kernel Model for Precision Timekeeping page. The presence of kernel support is automatically detected during the NTP build process and supporting code automatically compiled. In either case, the PPS signal must be present and within nominal jitter and wander tolerances. In addition, the prefer peer must be a truechimer; that is, survive the sanity checks and intersection algorithm. Finally, the offset of the system clock relative to the prefer peer must be within $\pm 0.5$ s. The kernel maintains a watchdog timer for the PPS signal; if the signal has not been heard or is out of tolerance for more than some interval, currently two minutes, the kernel discipline is disabled and operation continues as if it were not present.

An option flag in the driver determines whether the NTP algorithms or kernel support is enabled (if available). For historical reasons, the NTP algorithms are selected by default, since performance is generally better using older, slower systems. However, performance is generally better with kernel support using newer, faster systems.

8.7 Historical Notes on NTP Upgrades

This is an interim report on recent upgrades to the NTPv4 reference implementation code base and documentation. This report documents the upgrade program, which began in June 2007 and continued until March 2008. It is very important to recognize that this historic document describes the upgrade status as of 2008. Additional upgrades have been implemented since then. As of mid 2011, the additional upgrades are documented on the NTP Version 4 Release Notes page.

The motivation for this project was the overhaul and refinement of the code, some of which dates back twenty years. Some four dozen sets of fingers have introduced sometimes incompatible “improvements” that to some degree enhance or burden the product. There has been a continuing effort over the years to maintain the briar patch and pluck the more flagrant weeds, but it now requires a more systematic and thorough examination of purpose, design and implementation. The project is not complete, but far enough along to present a status report and review of significant changes.

Note: THE CHANGES DO NOT AFFECT THE PROTOCOL SPECIFICATION AND DO NOT AFFECT INTER-OPERABILITY WITH PREVIOUS VERSIONS.

8.7.1 1. Transparent Design

During the project a number of minor inconsistencies in various algorithms were found and resolved. In most cases this did not result in any changes in behavior, just a more simplified, transparent and easier to maintain design. In a few cases behavior has been modified to correct deficiencies and to avoid hostile attacks, as described below.

8.7.2 2. Documentation

The documentation required a major upgrade. Many pages have been overhauled, some completely rewritten and new ones added. A site map has been added and sorted by page category. A comprehensive command index has been added and sorted by page category. The command index includes a brief gloss for each command. A page has been added to show the various status word and event decodes used for monitoring and event reporting. The decodes show the internal code, ASCII report and short function gloss.

New pages have been added on association management, automatic server discovery and rate management. Much of the overburden on the program manual and configuration pages has been moved to these pages with the intent of the original pages to contain primarily a functional description for the commands and command line options. This is still an ongoing process.
8.7.3 3. Bulletproofing

In a continuing mission the code flow has been carefully adjusted to decrease vulnerability to configuration errors and possibly hostile attack. The order of restriction processing was adjusted to deflect access denials as early as possible and without consuming useless processor cycles. This is especially important in rate defense, as the MRU list should only be used for clients that could be legitimately served. In addition, the Autokey protocol was adjusted to avoid some potentially nasty disruption attacks.

8.7.4 4. Rate Management

Strict rate controls have been refined in both outbound and inbound traffic for both minimum headway (guard time) and minimum average headway. This is a major improvement over the original limireject design of 1992 and upgrade circa 2003. Headway violations result in an optional kiss-o’-death (KoD) packet. To avoid a clogging vulnerability, the KoD packets are themselves rate controlled for each source address separately.

The main feature of the revised design is that it is responsive to the server minimum headway and avoids guessing. This is done by setting the ppoll field in the server packet to the maximum of (a) the ppoll field in the client packet and (b) the server headway. The client sets the ppoll field in the association to the maximum of (a) the ppoll field in the server packet and (b) the minpoll field in the association. If this is a KoD and this value is greater than minpoll, minpoll is set to this value. The result is that the client continues sending, but only at headway at least as large as the server.

The revised design makes possible a decrease in the minimum time constant/poll interval to 3 (8 s), which reduces the risetime to 250 s. This may be useful for rapid convergence when a client is first started, but should not be used for links with moderate to large jitter. This is done using the average option of the discard command, which sets the minimum poll interval and headway from the default 4 (16 s) to a value in the range 3 (8 s) to 6 (64 s). Larger values than 4 might be appropriate for very busy public servers.

Rate management applies also to Autokey messages. This fixes a problem when iburst and autokey are both in play and when for some reason an association with iburst is repeatedly restarted. This may appear spooky to some folks that frequently restart a client for testing. The server remembers. Further information is in the current web documentation.

8.7.5 5. Frequency File

Initial frequency training has always been a problem, as it can take a very long time to trim the frequency estimate to nominal values. Once this happens and the frequency file is written, subsequent reboots will restore the frequency and frequency training is avoided. The problem is exacerbated using toll modem services such as ACTS which make a call at each poll interval. Until the training is complete the poll interval is held below the desired maximum as toll charges accrue.

The problem was solved by changing the clock state machine so that, if no frequency file is available, an initial training interval of 300 s occurs, after which the frequency is directly calculated and the discipline then turned over to the feedback loop. The choice of 300 s is based on the assumption that time can be estimated within 1 ms and the resulting frequency estimate within nominal 1 PPM.

Note that once the initial time offset is either stepped or slewed, no further time offsets are amortized during the training period. If the frequency error is large, the time offset at the end of the period can be moderately large, which then must be amortized by the feedback loop. While this may take up to an hour and result in a minor frequency tweak, the behavior is very much better than without the initial training. The remedy would require intricate and fragile code revisions.

In the original design the frequency file was written at one-hour intervals. This apparently makes embedded systems folks nervous, since this can tire the flash NVRAM after several years. The interval between writes now depends on the ambient clock stability and normally maxes out at something over one day unless the frequency takes an unusual twitch.

8.7. Historical Notes on NTP Upgrades
8.7.6 6. Leapseconds

The leapsecond processing has been overhauled once again. The problem is to avoid fake leap warnings displayed by an errant server and to insure correct response in case of large time changes which might validate or invalidate arming for a subsequent leap. No leap information is used unless the client is synchronized to a proveutic source. The values obtained from an Autokey server or peer are updated if newer than the current values. Server leap warning bits are disregarded if these values are available. If not, and if either a majority of the servers show leap warning bits or if one or more of the survivors are a reference clock with leap warning bit, the leap is armed. If armed by server leap warning bits and these provisions no longer prevail, the leap is disarmed. The NTPv4 protocol specifically does not speak to this issue.

The leap armed condition is displayed in the host status word. Transitions between warnings and no warnings are reported to the protostats file, system log and traps.

8.7.7 7. Orphan Mode and Local Clock Driver

The orphan mode code has been overhauled to correct some minor bugs and to clarify operation under normal and recovery conditions. The requirement that all subnet hosts have orphan configuration has been removed. The only requirement is that the orphan clients on the DMZ network sharing the root server(s) be so configured. The scheme now works if the root servers are configured with each other, either in symmetric or broadcast modes. Orphan mode is not considered in the NTPv4 protocol specification.

The local clock driver can be very dangerous when used as a fallback when connectivity to Internet time servers is interrupted. Orphan mode was designed to reduce the need for the local clock driver, as it is active only if no server is available. The local clock driver has been modified to have the same characteristics, regardless of stratum. Only if the host running the local clock driver loses all servers, regardless of stratum, is the driver activated. Thus, it is possible, but not recommended, to run the driver at any stratum, including zero.

8.7.8 8. Poll Rate Control

One of the most persistent problems is when after long operation and then a failure and then subsequently recovery, a client can take a long time to refresh the clock filter and resynchronize. Once the client has backed off the poll interval after a lengthy outage, it sends polls at that interval until receiving a response. At that time it temporarily retries at the minimum poll interval to fill up the clock filter. If iburst is configured, this will happen after 10 seconds or so and the client then resumes its poll interval required by the discipline time constant. This avoids needless network traffic while the poll interval increases gradually to the maximum. Further information is in the current web documentation.

The same thing happens on initial startup or when an association is restarted. The intent is to avoid a blast of iburst packets unless the server actually responds to the first one and to retry only while responding to the rate controls.

In order to speed response to initial startup when a reference clock is available, the clock is set on the first message received from the driver. This exposed an interesting bug, now fixed, with the ACTS modem driver, which began prematurely to ramp up the poll interval.

8.7.9 9. Autokey

The management of host and group names with respect to Autokey configuration and key generation has been removed and simplified. On host certificates, the subject and issuer fields carry the group name, while other certificates carry the host name, which can be an arbitrary string having nothing to do with the DNS name. This opens up a possible future plan to use the Autokey name rather than the IP address when constructing the session key. It also allows a client to easily switch from one group to another without regenerating the certificate. Further information is in the current web documentation and in the latest Autokey ID.
Various protocol refinements have been done in the Autokey state machine. A bug was found in symmetric modes where the peer cookies were not EXORed. A bug was found in processing the certificate cache when a participant was a client of two or more server in the same group which themselves had certificate trails to different trusted hosts.

The protocol machine is now restarted every several days in order to update certificates and leapseconds values when they are changed.

### 8.7.10 10. Report, Log and Event Codes

The status, selection, source, event and log decodes have been adjusted for consistency. Some of the decodes were missing, some with errors and a few new ones added. Old versions of ntpq continue to work without change, but display a new code as space. Except for the new codes, this behavior is consistent with RFC 1305 and proposed for the NTPv4 protocol specification.

The ntpq as command has been changed to fix some very old bugs. The display is now consistent with the system and peer billboards. The authentication state is correctly displayed for broadcast server associations.

The event reporting has been cleaned up for more straightforward interpretation by a remote agent. All significant state transitions are reported, including clock state machine changes, mobilization, /demobilization, system and peer restart, system peer change, panic stop and so forth.

A new protostats monitoring file facility has been added. It works just like the other monitor files. All events are recorded to this file as reported and optionally to the system log. Many reports that sometimes clog up the system log are more usefully directed to this file. The reports also trigger a trap packet that can be sent via an agent to page an administrator.

When the current mode-6 monitoring protocol was designed circa 1988 the considered intent was that monitoring functions rely only on the NTP packet itself and the system, peer and clock status words provided in the mode-6 packet. While the strongly felt advice at that time was to avoid reformatting the plain ASCII text sent by the server, at various times folks have cheated and reformatted the text. In some places this is good, like displaying the filter shift register; in some places this is bad, like reformatting the timestamps. There is nothing much that can be done about this now without angry mobs rioting when forced to upgrade to a new ntpq. I will not rule this out in future.

A more serious comment has to do with using other than the NTP packet, status words and events for monitoring purposes. Emphasis added: monitors should not parse such things as the flash codes, clock state or anything else not called out in the NTPv4 specification. The clock state machine is defined in the specification, but no specific numbers are assigned to the states.

When the numbers were changed to align for reporting purposes, some scripts no longer worked. The scripts should be changed to use only the leap and select fields of the system status word. If the leap field is other than 0, the client has synchronized at least once; if the select field is other than 0, the client is currently synchronized to the source indicated in the decode.

### 8.7.11 11. Two-step and timestamp capture

A number of interesting ideas were found in the IEEE 1588 Precision Time Protocol specification. One of them was the two-step protocol in which the transmit timestamp is sent in a following message. However, the PTP design operates only in a master-slave configuration and is not directly usable in NTP. The protocol was adapted to the NTP symmetric design, which requires four state variables rather than two. It is described on Timestamp Capture Principles. This might be an interesting project for future research.

A detailed study of the timestamp capture opportunities for both hardware and software timestamping revealed that the most accurate and interoperable design involves the transmit timestamp at the beginning of the packet and then receive timestamp at the end. This makes it possible to accurately measure the offset and delay even if the ends of the synchronization path operate at different rates. It is described on the Timestamp Capture Principles page.
8.7.12 12. Windows client bug

The Windows XP and Vista clients send the NTP request in symmetric active mode rather than client mode. An unsuspecting server could mobilize a symmetric passive association, which is a serious security vulnerability. The NTPv4 servers, including those at NIST and USNO, discard symmetric active requests unless cryptographically authenticated, so Windows clients do not work. The Microsoft KB 875424 discusses the preferred workaround; however, an optional workaround is now available so that, if the request is not authenticated, the server responds with symmetric passive mode, but without mobilize an association. The workaround is enabled with the WINTIME build option.

The spec assumes that either peer in symmetric modes can synchronize the other should a peer lose all sources. The workaround violates that assumption and some legitimate configuration might be badly misused. It should be used only with this understanding.

8.7.13 13. Autonomous configuration

The autonomous configuration (pool and manycast) code was refined to more reliably prune excess servers. If a truechimer is discarded by the clustering algorithm and the total number of survivors is greater than the maxclock option of the tos command, it is considered excess and shows a “#” tally code. If the association is ephemeral and survives the clustering algorithm, the watchdog counter is reset. If the watchdog timer expires and the total number of associations is greater than the maxclock option of the tos command, it is demobilized. This behavior is not considered in the NTPv4 protocol specification.

8.7.14 14. Code ornamentation

When auditing the code and figuring out its historic origin and evolution, additional commentary has been added so future generations can figure it out, too.

David L. Mills
17 March 2008

8.8 Copyright Notice

The following copyright notice applies to all files collectively called the Network Time Protocol Version 4 Distribution. Unless specifically declared otherwise in an individual file, this entire notice applies as if the text was explicitly included in the file.

***********************************************************************
* Copyright (c) University of Delaware 1992-2015                        *
* Permission to use, copy, modify, and distribute this software and     *
* its documentation for any purpose with or without fee is hereby      *
* granted, provided that the above copyright notice appears in all     *
* copies and that both the copyright notice and this permission       *
* notice appear in supporting documentation, and that the name         *
* University of Delaware not be used in advertising or publicity       *
* pertaining to distribution of the software without specific,         *
* written prior permission. The University of Delaware makes no        *
* representations about the suitability this software for any           *
* purpose. It is provided "as is" without express or implied          *
* warranty.                                                           *
***********************************************************************

134 Chapter 8. Miscellaneous Pages
The following individuals contributed in part to the Network Time Protocol Distribution Version 4 and are acknowledged as authors of this work.

1. Takao Abe <takao_abe@xurb.jp> Clock driver for JJY receivers
2. Mark Andrews <mark_andrews@isc.org> Leitch atomic clock controller
3. Bernd Altmeier <altmeier@atlsoft.de> hopf Elektronik serial line and PCI-bus devices
4. Viraj Bais <vbais@mailman1.intel.com> and Clayton Kirkwood <kirkwood@striderfm.intel.com> port to WindowsNT 3.5
5. Michael Barone <michael.barone@lmco.com> GPSVME fixes
6. Karl Berry <karl@owl.HQ.ileaf.com> syslog to file option
7. Greg Brackley <greg.brackley@bigfoot.com> Major rework of WINNT port. Clean up recvbuf and iosignal code into separate modules.
8. Marc Brett <Marc.Brett@westgeo.com> Magnavox GPS clock driver
9. Piete Brooks <Piete.Brooks@cl.cam.ac.uk> MSF clock driver, Trimble PARSE support
10. Nelson B Bolyard <nelson@bolyard.me> update and complete broadcast and crypto features in sntp
11. Jean-Francois Boudreault <Jean-Francois.Boudreault@viagenie.qc.ca> IPv6 support
12. Reg Clemens <reg@dwf.com> Oncore driver (Current maintainer)
13. Steve Clift <clift@ml.csiro.au> OMEGA clock driver
14. Casey Crellin <casey@csc.co.za> vxWorks (Tornado) port and help with target configuration
15. Sven Dietrich <sven_dietrich@trimble.com> Palisade reference clock driver, NT adj. residuals, integrated Greg’s Winnt port.
16. John A. Dundas III <dundas@salt.jpl.nasa.gov> Apple A/UX port
17. Torsten Duwe <duwe@immd4.informatik.uni-erlangen.de> Linux port
18. Dennis Ferguson <dennis@mrbill.canet.ca> foundation code for NTP Version 2 as specified in RFC 1119
19. John Hay <jhay@icomtek.csir.co.za> IPv6 support and testing
20. Dave Hart <davehart@davehart.com> General maintenance, Windows port interpolation rewrite
21. Claas Hilbrecht <neoclock4x@linum.com> NeoClock4X clock driver
22. Glenn Hollinger <glenn@herald.usask.ca> GOES clock driver
23. Mike Iglesias <iglesias@uci.edu> DEC Alpha port
24. Jim Jaglielski <jim@jagubox.gsfc.nasa.gov> A/UX port
25. Jeff Johnson <jbj@chatham.usdesign.com> massive prototyping overhaul
26. Hans Lambermont <Hans.Lambermont@nl.origin-it.com> or <H.Lambermont@chello.nl> ntpswEEP
27. Poul-Henning Kamp <phk@FreeBSD.ORG> Oncore driver (Original author)
28. Frank Kardel <kardel (at) ntp (dot) org> PARSE <GENERIC> (driver 14 reference clocks), STREAMS modules for PARSE, support scripts, syslog cleanup, dynamic interface handling
29. Johannes Maximilian Kuehn <kuehn@ntp.org> Rewrote snTP to comply with NTPv4 specification, ntpq saveconfig
30. William L. Jones <jones@hermes.chpc.utexas.edu> RS/6000 AIX modifications, HPUX modifications
31. Dave Katz <dkatz@cisco.com> RS/6000 AIX port
32. Craig Leres <leres@ee.lbl.gov> 4.4BSD port, ppsclock, Magnavox GPS clock driver
33. George Lindholm <lindholm@ucs.ubc.ca> SunOS 5.1 port
34. Louis A. Mamakos <louie@ni.umd.edu> MD5-based authentication
35. Lars H. Mathiesen <thorinn@diku.dk> adaptation of foundation code for Version 3 as specified in RFC 1305
36. Danny Mayer <mayer@ntp.org> Network I/O, Windows Port, Code Maintenance
37. David L. Mills <mills@udel.edu> Version 4 foundation, precision kernel; clock drivers: 1, 3, 4, 6, 7, 11, 13, 18, 19, 22, 36
38. Wolfgang Moeller <moeller@gwdgv1.dnet.gwdg.de> VMS port
39. Jeffrey Mogul <mogul@pa.dec.com> ntptrace utility
40. Tom Moore <tmoore@fievel.daytonoh.ncr.com> i386 svr4 port
41. Kamal A Mostafa <kamal@whence.com> SCO OpenServer port
42. Derek Mulcahy <derek@toybox.demon.co.uk> and Damon Hart-Davis <d@hd.org> ARCRON MSF clock driver
43. Rob Neal <neal@ntp.org> Bancomm refclock and config/parse code maintenance
44. Rainer Pruy <Rainer.Pruy@informatik.uni-erlangen.de> monitoring/trap scripts, statistics file handling
45. Dirce Richards <dirce@zk3.dec.com> Digital UNIX V4.0 port
46. Wilfredo Sánchez <wsanchez@apple.com> added support for NetInfo
47. Nick Sayer <mrapple@quack.kfu.com> SunOS streams modules
48. Jack Sasportas <jack@innovativeinternet.com> Saved a Lot of space on the stuff in the html/pic/ subdirectory
49. Ray Schnitzler <schnitz@unipress.com> Unixware1 port
50. Michael Shields <shields@tembel.org> USNO clock driver
51. Jeff Steinman <jss@pebbles.jpl.nasa.gov> Datum PTS clock driver
52. Harlan Stenn <harlan@pfcs.com> GNU automake/autoconfigure makeover, various other bits (see the ChangeLog)
53. Kenneth Stone <ken@sdd.hp.com> HP-UX port
54. Ajit Thyagarajan <ajit@ee.udel.edu> IP multicast/anycast support
55. Tomoaki TSURUOKA <tsuruoka@nc.fukuoka-u.ac.jp> TRAK clock driver
56. Brian Utterback <brian.utterback@oracle.com> General codebase, Solaris issues
57. Loganaden Velvindron <loganaden@gmail.com> Sandboxing (libseccomp) support
58. Paul A Vixie <vixie@vix.com> TrueTime GPS driver, generic TrueTime clock driver
59. Ulrich Windl <Ulrich.Windl@rz.uni-regensburg.de> corrected and validated HTML documents according to the HTML DTD

8.8. Copyright Notice
9.1 How NTP Works

9.1.1 Abstract

This page and its dependencies contain a technical description of the Network Time Protocol (NTP) architecture and operation. It is intended for administrators, operators and monitoring personnel. Additional information for nontechnical readers can be found in the white paper Executive Summary: Computer Network Time Synchronization. While this page and its dependencies are primarily concerned with NTP, additional information on related protocols can be found in the white papers IEEE 1588 Precision Time Protocol (PTP) and Time Synchronization for Space Data Links. Note that reference to a page in this document collection is to a page in the collection, while reference to a white paper is to a document at the Network Time Synchronization Research Project web site.

9.1.2 Introduction and Overview

NTP time synchronization services are widely available in the public Internet. The public NTP subnet currently includes several thousand servers in most countries and on every continent of the globe, including Antarctica, and sometimes in space and on the sea floor. These servers support, directly or indirectly, a total population estimated at over 25 million computers in the global Internet.

The NTP subnet operates with a hierarchy of levels, where each level is assigned a number called the stratum. Stratum 1 (primary) servers at the lowest level are directly synchronized to national time services via satellite, radio or telephone modem. Stratum 2 (secondary) servers at the next higher level are synchronized to stratum 1 servers and so on. Normally, NTP clients and servers with a relatively small number of clients do not synchronize to public primary servers. There are several hundred public secondary servers operating at higher strata and are the preferred choice.

This page presents an overview of the NTP implementation included in this software distribution. We refer to this implementation as the reference implementation only because it was used to test and validate the NTPv4 specification RFC 5905. It is best read in conjunction with the briefings and white papers on the Network Time Synchronization Research Project page. An executive summary suitable for management and planning purposes is in the white paper Executive Summary: Computer Network Time Synchronization.

9.1.3 NTP Timescale and Data Formats

NTP clients and servers synchronize to the Coordinated Universal Time (UTC) timescale used by national laboratories and disseminated by radio, satellite and telephone modem. This is a global timescale independent of geographic position. There are no provisions to correct for local time zone or daylight savings time; however, these functions can be performed by the operating system on a per-user basis.
The UT1 timescale, upon which UTC is based, is determined by the rotation of the Earth about its axis. The Earth rotation is gradually slowing down relative to International Atomic Time (TAI). In order to rationalize UTC with respect to TAI, a leap second is inserted at intervals of about 18 months, as determined by the International Earth Rotation Service (IERS). Reckoning with leap seconds in the NTP timescale is described in the white paper The NTP Timescale and Leap Seconds.

The historic insertions are documented in the leap-seconds.list file, which can be downloaded from the NIST FTP servers. This file is updated at intervals not exceeding six months. Leap second warnings are disseminated by the national laboratories in the broadcast timecode format. These warnings are propagated from the NTP primary servers via other server to the clients by the NTP on-wire protocol. The leap second is implemented by the operating system kernel, as described in the white paper The NTP Timescale and Leap Seconds. Implementation details are described on the Leap Second Processing page.

Figure 9.1: NTP Data Formats

Figure 9.1 shows two NTP time formats, a 64-bit timestamp format and a 128-bit datestamp format. The datestamp format is used internally, while the timestamp format is used in packet headers exchanged between clients and servers. The timestamp format spans 136 years, called an era. The current era began on 1 January 1900, while the next one begins in 2036. Details on these formats and conversions between them are in the white paper The NTP Era and Era Numbering. However, the NTP protocol will synchronize correctly, regardless of era, as long as the system clock is set initially within 68 years of the correct time. Further discussion on this issue is in the white paper NTP Timestamp Calculations. Ordinarily, these formats are not seen by application programs, which convert these NTP formats to native Unix or Windows formats.

9.1.4 Architecture and Algorithms

The overall organization of the NTP architecture is shown in Figure 9.2. It is useful in this context to consider the implementation as both a client of upstream (lower stratum) servers and as a server for downstream (higher stratum)
clients. It includes a pair of peer/poll processes for each reference clock or remote server used as a synchronization source. Packets are exchanged between the client and server using the on-wire protocol described in the white paper Analysis and Simulation of the NTP On-Wire Protocols. The protocol is resistant to lost, replayed or spoofed packets.

The poll process sends NTP packets at intervals ranging from 8 s to 36 hr. The intervals are managed as described on the Poll Process page to maximize accuracy while minimizing network load. The peer process receives NTP packets and performs the packet sanity tests described on the Event Messages and Status Words page and flash status word. The flash status word reports in addition the results of various access control and security checks described in the white paper NTP Security Analysis. A sophisticated traffic monitoring facility described on the Rate Management and the Kiss-o’-Death Packet page protects against denial-of-service (DoS) attacks.

Packets that fail one or more of these tests are summarily discarded. Otherwise, the peer process runs the on-wire protocol that uses four raw timestamps: the origin timestamp $T_1$ upon departure of the client request, the receive timestamp $T_2$ upon arrival at the server, the transmit timestamp $T_3$ upon departure of the server reply, and the destination timestamp $T_4$ upon arrival at the client. These timestamps, which are recorded by the rawstats option of the filegen command, are used to calculate the clock offset and roundtrip delay samples:

$$\text{offset} = \frac{[T_2 - T_1] + (T_3 - T_4)}{2},$$

$$\text{delay} = (T_4 - T_1) - (T_3 - T_2).$$

In this description the transmit timestamps $T_1$ and $T_3$ are softstamps measured by the inline code. Softstamps are subject to various queuing and processing delays. A more accurate measurement uses drivestamps, as described on the NTP Interleaved Modes page. These issues along with mathematical models are discussed in the white paper NTP Timestamp Calculations.

The offset and delay statistics for one or more peer processes are processed by a suite of mitigation algorithms. The algorithm described on the Clock Filter Algorithm page selects the offset and delay samples most likely to produce accurate results. Those servers that have passed the sanity tests are declared selectable. From the selectable population the statistics are used by the algorithm described on the Clock Select Algorithm page to determine a number of truechimers according to Byzantine agreement and correctness principles. From the truechimer population the algorithm described on the Clock Cluster Algorithm page determines a number of survivors on the basis of statistical clustering principles.

The algorithms described on the Mitigation Rules and the prefer Keyword page combine the survivor offsets, designate one of them as the system peer and produces the final offset used by the algorithm described on the Clock Discipline Algorithm page to adjust the system clock time and frequency. The clock offset and frequency, are recorded by the loopstats option of the filegen command. For additional details about these algorithms, see the Architecture Briefing on the Network Time Synchronization Research Project page. For additional information on statistical principles and performance metrics, see the Performance Metrics page.

### 9.2 Mitigation Rules and the prefer Keyword

This page summarizes the criteria for choosing from among the survivors of the clock cluster algorithm a set of contributors to the clock discipline algorithm. The criteria are very meticulous, since they have to handle many different scenarios that may be optimized for special circumstances, including some scenarios designed to support planetary and deep space missions. For additional information on statistical principles and performance metrics, see the Performance Metrics page.

Recall the suite of NTP data acquisition and grooming algorithms. These algorithms proceed in five phases. Phase one discovers the available sources and mobilizes an association for each source found. These sources can result from explicit configuration, broadcast discovery or the pool and manycast autonomous configuration schemes. See the Automatic Server Discovery Schemes page for further information.

Phase two selects the candidates from among the sources by excluding those sources showing one or more of the errors summarized on the Clock Select Algorithm page and to determine the truechimers from among the candidates, leaving behind the falsetickers. A server or peer configured with the true option is declared a truechimer independent
of this algorithm. Phase four uses the algorithm described on the Clock Cluster Algorithm page to prune the statistical outliers from the truechimers, leaving the survivor list as result.

Phase five uses a set of algorithms and mitigation rules to combined the survivor statistics and discipline the system clock. The mitigation rules select from among the survivors a system peer from which a set of system statistics can be inherited and passed along to dependent clients, if any. The mitigation algorithms and rules are the main topic of this page. The clock offset developed from these algorithms can discipline the system clock, either using the clock discipline algorithm or using the kernel to discipline the system clock directly, as described on the A Kernel Model for Precision Timekeeping page.

9.2.1 Combine Algorithm

The clock combine algorithm uses the survivor list to produce a weighted average of both offset and jitter. Absent other considerations discussed later, the combined offset is used to discipline the system clock, while the combined jitter is augmented with other components to produce the system jitter statistic inherited by dependent clients, if any.

The clock combine algorithm uses a weight factor for each survivor equal to the reciprocal of the root distance. This is normalized so that the sum of the reciprocals is equal to unity. This design favors the survivors at the smallest root distance and thus the smallest maximum error.

9.2.2 Anti-Clockhop Algorithm

The anti-clockhop algorithm is intended for cases where multiple servers are available on a fast LAN with modern computers. Typical offset differences between servers in such cases are less than 0.5 ms. However, changes between servers can result in unnecessary system jitter. The object of the anti-clockhop algorithm is to avoid changing the current system peer, unless it becomes stale or has significant offset relative to other candidates on the survivor list.

For the purposes of the following description, call the last selected system peer the old peer, and the currently selected source the candidate peer. At each update, the candidate peer is selected as the first peer on the survivor list sorted by increasing root distance. The algorithm initializes the -clockhop threshold with the value of mindist, by default 1 ms.

The anti-clockhop algorithm is called immediately after the combine algorithm. If there was no old peer or the old and candidate peers are the same, the candidate peer becomes the system peer. If the old peer and the candidate peer are different, the algorithm measures the difference between the offset of the old peer and the candidate peer. If the difference exceeds the clockhop threshold, the candidate peer becomes the system peer and the clockhop threshold is restored to its original value. If the difference is less than the clockhop threshold, the old peer continues as the system peer. However, at each subsequent update, the algorithm reduces the clockhop threshold by half. Should operation continue in this way, the candidate peer will eventually become the system peer.

9.2.3 Peer Classification

The behavior of the various algorithms and mitigation rules involved depends on how the various synchronization sources are classified. This depends on whether the source is local or remote and if local, the type of source. The following classes are defined:

1. A selectable association configured for a remote server or peer is classified as a client association. All other selectable associations are classified as device driver associations of one kind or another. In general, one or more sources of either type will be available in each installation.

2. If all sources have been lost and one or more hosts on a common DMZ network have specified the orphan stratum in the orphan option of the tos command, each of them can become an orphan parent. Dependent orphan children on the same DMZ network will see the orphan parents as if synchronized to a server at the
orphan stratum. Note that, as described on the Orphan Mode page, all orphan children will select the same orphan parent for synchronization.

3. When a device driver has been configured for pulse-per-second (PPS) signals and PPS signals are being received, it is designated the PPS driver. Note that the Pulse-per-Second driver (type 22) is often used as a PPS driver, but any driver can be configure as a PPS driver if the hardware facilities are available. The PPS driver provides precision clock discipline only within $\pm 0.4$ s, so it is always associated with another source or sources that provide the seconds numbering function.

4. When the Undisciplined Local Clock driver (type 1) is configured, it is designated the local driver. It can be used either as a backup source (stratum greater than zero) should all sources fail, or as the primary source (stratum zero) whether or not other sources are available if the prefer option is present. The local driver can be used when the kernel time is disciplined by some other means of synchronization, such as the NIST lock clock scheme, or another synchronization protocol such as the IEEE 1588 Precision Time Protocol (PTP) or Digital Time Synchronization Service (DTSS).

5. When the Automated Computer Time Service driver (type 18) is configured, it is designated the modem driver. It is used either as a backup source, should all other sources fail, or as the primary source if the prefer option is present.

9.2.4 The prefer Peer

The mitigation rules are designed to provide an intelligent selection of the system peer from among the selectable sources of different types. When used with the server or peer commands, the prefer option designates one or more sources as preferred over all others. While the rules do not forbid it, it is usually not useful to designate more than one source as preferred; however, if more than one source is so designated, they are used in the order specified in the configuration file. If the first one becomes un selectable, the second one is considered and so forth. This order of priority is also applicable to multiple PPS drivers, multiple modem drivers and even multiple local drivers, although that would not normally be useful.

The cluster algorithm works on the set of truechimers produced by the select algorithm. At each round the algorithm casts off the survivor least likely to influence the choice of system peer. If selectable, the prefer peer is never discarded; on the contrary, its potential removal becomes a termination condition. However, the prefer peer can still be discarded by the select algorithm as a falseticker; otherwise, the prefer peer becomes the system peer.

Ordinarily, the combine algorithm computes a weighted average of the survivor offset and jitter to produce the final values. However, if a prefer peer is among the survivors, the combine algorithm is not used. Instead, the offset and jitter of the prefer peer are used exclusively as the final values. In the common case involving a radio clock and a flock of remote backup servers, and with the radio clock designated a prefer peer, the radio clock disciplines the system clock as long as the radio itself remains operational. However, if the radio fails or becomes a falseticker, the combined backup sources continue to discipline the system clock.

9.2.5 Mitigation Rules

As the select algorithm scans the associations for selectable candidates, the modem driver and local driver are segregated for later, but only if not designated a prefer peer. If so designated, the driver is included among the candidate population. In addition, if orphan parents are found, the parent with the lowest metric is segregated for later; the others are discarded. For this purpose the metric is defined as the four-octet IPv4 address or the first four octets of the hashed IPv6 address. The resulting candidates, including any prefer peers found, are processed by the select algorithm to produce a possibly empty set of truechimers.

As previously noted, the cluster algorithm casts out outliers, leaving the survivor list for later processing. The survivor list is then sorted by increasing root distance and the first entry temporarily designated the system peer. At this point the following contributors to the system clock discipline may be available:

• (potential) system peer, if there are survivors;
The mitigation algorithm proceeds in three steps in turn.

1. If there are no survivors, the modem driver becomes the only survivor if there is one. If not, the local driver becomes the only survivor if there is one. If not, the orphan parent becomes the only survivor if there is one. If the number of survivors at this point is less than the `minsane` option of the `tos` command, the algorithm is terminated and the system variables remain unchanged. Note that `minsane` is by default 1, but can be set at any value including 0.

2. If the prefer peer is among the survivors, it becomes the system peer and its offset and jitter are inherited by the corresponding system variables. Otherwise, the combine algorithm computes these variables from the survivor population.

3. If there is a PPS driver and the system clock offset at this point is less than 0.4 s, and if there is a prefer peer among the survivors or if the PPS peer is designated as a prefer peer, the PPS driver becomes the system peer and its offset and jitter are inherited by the system variables, thus overriding any variables already computed. Note that a PPS driver is present only if PPS signals are actually being received and enabled by the associated driver.

If none of the above is the case, the data are disregarded and the system variables remain as they are.

### 9.2.6 The `minsane` Option

The `minsane` option of the `tos` command, the `prefer` option of the `server` and `peer` commands and the `flag` option of the `fudge` command for a selected driver can be used with the mitigation rules to provide many useful configurations. The `minsane` option specifies the minimum number of survivors required to synchronize the system clock. The `prefer` option operates as described in previous sections. The `flag` option enables the PPS signal for the selected driver.

A common scenario is a GPS driver with a serial timecode and PPS signal. The PPS signal is disabled until the system clock has been set by some means, not necessarily the GPS driver. If the serial timecode is within 0.4 s of the PPS signal, the GPS driver is designated the PPS driver and the PPS signal disciplines the system clock. If the serial timecode becomes unreliable, or if the PPS signal is disconnected, the GPS driver stops updating the system clock and so eventually becomes unreachable and is replaced by other sources.

Whether or not the GPS driver disables the PPS signal when the timecode becomes unreliable is at the discretion of the driver. Ordinarily, the PPS signal is disabled in this case; however, when the GPS receiver has a precision holdover oscillator, the driver may elect to continue PPS discipline. In this case, `minsane` can be set to zero so the PPS signal continues to discipline the system clock.

### 9.3 Autokey Public-Key Authentication

This distribution includes support for the Autokey public key algorithms and protocol specified in RFC 5906 “Network Time Protocol Version 4: Autokey Specification”. This support is available only if the OpenSSL library has been installed and the `--enable-autokey` option is specified when the distribution is built.

Public key cryptography is generally considered more secure than symmetric key cryptography. Symmetric key cryptography is based on a shared secret key which must be distributed by secure means to all participants. Public key
cryptography is based on a private secret key known only to the originator and a public key known to all participants. A recipient can verify the originator has the correct private key using the public key and any of several digital signature algorithms.

The Autokey Version 2 protocol described on the Autokey Protocol page verifies packet integrity using message digest algorithms, such as MD5 or SHA, and verifies the source using digital signature schemes, such as RSA or DSA. As used in Autokey, message digests are exceptionally difficult to cryptanalyze, as the keys are used only once.

Optional identity schemes described on the Autokey Identity Schemes page are based on cryptographic challenge/response exchanges. Optional identity schemes provide strong security against masquerade and most forms of clogging attacks. These schemes are exceptionally difficult to cryptanalyze, as the challenge/response exchange data are used only once. They are described along with an executive summary, current status, briefing slides and reading list on the Autonomous Authentication page.

Autokey authenticates individual packets using cookies bound to the IP source and destination addresses. The cookies must have the same IP addresses at both the server and client. For this reason operation with network address translation schemes is not possible. This reflects the intended robust security model where government and corporate NTP servers and clients are operated outside firewall perimeters.

Autokey is designed to authenticate servers to clients, not the other way around as in SSH. An Autokey server can support an authentication scheme such as the Trusted Certificate (TC) scheme described in RFC 5906, while a client is free to choose between the various options. It is important to understand that these provisions are optional and that selection of which option is at the discretion of the client. If the client does not require authentication, it is free to ignore it, even if some other client of the same server elects to participate in either symmetric key or public key cryptography.

Autokey uses industry standard X.509 public certificates, which can be produced by commercial services, utility programs in the OpenSSL software library, and the ntp-keygen utility program in the NTP software distribution. A certificate includes the subject name of the client, the issuer name of the server, the public key of the client and the time period over which the the public and private keys are valid. All Autokey hosts have a self-signed certificate with the Autokey name as both the subject and issuer. During the protocol, additional certificates are produced with the Autokey host name as subject and the host that signs the certificate as issuer.

There are two timeouts associated with the Autokey scheme. The key list timeout is set by the automax command, which specifies the interval between generating new key lists by the client or server. The default timeout of about 1.1 hr is appropriate for the majority of configurations and ordinarily should not be changed. The revoke timeout is set by the revoke command, which specifies the interval between generating new server private values. It is intended to reduce the vulnerability to cryptanalysis; however, new values require the server to encrypt each client cookie separately. The default timeout of about 36 hr is appropriate for most servers, but might be too short for national time servers.

### 9.3.1 Autokey Subnets

An Autokey subnet consists of a collection of hosts configured as an acyclic, directed tree with roots one or more trusted hosts (THs) operating at the lowest stratum of the subnet. Note that the requirement that the NTP subnet be acyclic means that, if two hosts are configured with each other in symmetric modes, each must be a TH. The THs are synchronized directly or indirectly to national time services via trusted means, such as radio, satellite or telephone modem, or one or more trusted agents (TAs) of a parent subnet. NTP subnets can be nested, with the THs of a child subnet configured for one or more TAs of a parent subnet. The TAs can serve one or more child subnets, each with its own security policy and set of THs.

A certificate trail is a sequence of certificates, each signed by a host one step closer to the THs and terminating at the self-signed certificate of a TH. The requirement that the subnet be acyclic means certificate trails can never loop. NTP servers operate as certificate authorities (CAs) to sign certificates provided by their clients. The CAs include the TAs of the parent subnet and those subnet servers with dependent clients.

In order for the signature to succeed, the client certificate valid period must begin within the valid period of the server certificate. If the server period begins later than the client period, the client certificate has expired; if the client period
begins later than the server period, the server certificate has expired.

The Autokey protocol runs for each association separately. During the protocol, the client recursively obtains the certificates on the trail to a TH, saving each in a cache ordered from most recent to oldest. If an expired certificate is found, it is invalidated and marked for later replacement. As the client certificate itself is not involved in the certificate trail, it can only be declared valid or expired when the server signs it.

The certificates derived from each association are combined in the cache with duplicates suppressed. If it happens that two different associations contribute certificates to the cache, a certificate on the trail from one association could expire before any on another trail. In this case the remaining trails will survive until the expired certificate is replaced. Once saved in the cache, a certificate remains valid until it expires or is replaced by a new one.

It is important to note that the certificate trail is validated only at startup when an association is mobilized. Once validated in this way, the server remains valid until it is demobilized, even if certificates on the trail to the THs expire. While the certificate trail authenticates each host on the trail to the THs, it does not validate the time values themselves. Ultimately, this is determined by the NTP on-wire protocol.

Example

![Figure 9.3: Example Configuration](image)

Figure 9.3 shows an example configuration with three NTP subnets, Alice, Helen and Carol. Alice and Helen are parent groups for Carol with TA C belonging to Alice and TA S belonging to Helen. Hosts A and B are THs of Alice, host R is the TH of Helen and host X is the TH of Carol. Assume that all associations are client/server, child subnet TH X has two mobilized associations, one to Alice TA host C and the other to Carol TA host S. While not shown in the figure, Alice hosts A and B could configure symmetric mode associations between them for redundancy and backup.

Note that host D certificate trail is D→C→A or D→C→B, depending on the particular order the trails are built. Host Y certificate trail is only Y→X, since X is a TH. Host X has two certificate trails X→C→A or X→C→B, and X→S→R.

### 9.3.2 Subnet Group Names

In some configurations where more than one subnet shares an Ethernet or when multiple subnets exist in a manycast or pool configuration, it is useful to isolate one subnet from another. In Autokey this can be done using group names. An Autokey host name is specified by the `-s <host>@<group>` option of the `ntp-keygen` program, where `<host>` is the host name and `<group>` is the group name. If `<host>` is omitted, the name defaults to the string returned by the Unix `gethostname()` routine, ordinarily the DNS name of the host. Thus, for host `beauregard.udel.edu` the option `-s @red` specifies the Autokey host name `beauregard.udel.edu@red`.

A subnet host with a given group name will discard ASSOC packets from all subnets with a different group name. This effectively disables the Autokey protocol without additional packet overhead. For instance, one or more manycast or pool servers will not respond to ASSOC packets from subnets with difference group names. Groups sharing an Ethernet will be filtered in the same way.
However, as shown in Figure 9.3, there are configurations where a TH of one group needs to listen to a TA of a different group. This is accomplished using the `ident group` option of the `crypto` command and/or the `ident group` option of the `server` command. The former case applies to all hosts sharing a common broadcast, manycast or symmetric passive modes, while the latter case applies to each individual client/server or symmetric active mode association. In either case the host listens to the specified group name in addition to the group name specified in the `-s` option of the `ntp-keygen` program.

### 9.3.3 Secure Groups

NTP security groups are an extension of the NTP subnets described in the previous section. They include in addition to certificate trails one or another identity schemes described on the Autokey Identity Schemes page. NTP secure groups are used to define cryptographic compartments and security hierarchies. The identity scheme insures that the server is authentic and not victim of masquerade by an intruder acting as a middleman.

An NTP secure group is an NTP subnet configured as an acyclic tree rooted on the THs. The THs are at the lowest stratum of the secure group. They run an identity exchange with the TAs of parent subnets. All group hosts construct an unbroken certificate trail from each host, possibly via intermediate hosts, and ending at a TH of that group. The TH verifies authenticity with the TA of the parent subnet using an identity exchange.

![Figure 9.4: Identify Scheme](image)

The identity exchange is run between a TA acting as a server and a TH acting as a client. As shown in Figure 9.4, the identity exchange involves a challenge-response protocol where a client generates a nonce and sends it to the server. The server performs a mathematical operation involving a second nonce and the secret group key, and sends the result along with a hash to the client. The client performs another mathematical operation and verifies the result with the hash.

Since each exchange involves two nonces, even after repeated observations of many exchanges, an intruder cannot learn the secret group key. It is this quality that allows the secret group key to persist long after the longest period of certificate validity. In the Schnorr (Identify Friend or Foe - IFF) scheme, the secret group key is not divulged to the clients, so they cannot conspire to prove identity to other hosts.

As described on the Autokey Identity Schemes page, there are five identity schemes, three of which - IFF, GQ and MV - require identity files specific to each scheme. There are two types of files for each scheme, an encrypted server keys file and a nonencrypted client keys file, also called the parameters file, which usually contains a subset of the keys file.

Figure 9.4 shows how keys and parameters are distributed to servers and clients. A TA constructs the encrypted keys file and the nonencrypted parameters file. Hosts with no dependent clients can retrieve client parameter files from an archive or web page. The `ntp-keygen` program can export parameter files using the `-e` option. By convention, the file name is the name of the secure group and must match the `ident` option of the `crypto` command or the `ident` option of the `server` command.
When more than one TH Is involved in the secure group, it is convenient for the TAs and THs to use the same encrypted key files. To do this, one of the parent TAs includes the \(-i <\text{group}>\) option on the \texttt{ntp-keygen} command line, where \(<\text{group}>\) is the name of the child secure group. The \texttt{ntp-keygen} program can export server keys files using the \(-q\) option and a chosen remote password. The files are installed on the TAs and then renamed using the name given as the first line in the file, but without the filestamp. The secure group name must match the \texttt{ident} option for all TAs.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Caution:} & In the latest Autokey version, the host name and group name are independent of each other and the \texttt{host} option of the \texttt{crypto} command is deprecated. When compatibility with older versions is required, specify the same name for both the \(-s\) and \(-i\) options. \\
\hline
\end{tabular}
\end{table}

In special circumstances the Autokey message digest algorithm can be changed using the \texttt{digest} option of the \texttt{crypto} command. The digest algorithm is separate and distinct from the symmetric key message digest algorithm. If compliance with FIPS 140-2 is required, the algorithm must be either SHA or SHA1. The Autokey message digest algorithm must be the same for all participants in the NTP subnet.

\textbf{Example}

Returning to the example of Figure 9.3, Alice, Helen and Carol run run the Trusted Certificate (TC) scheme, internally, as the environment is secure and without threat from external attack, in particular a middleman masquerade. However, TH X of Carol is vulnerable to masquerade on the links between X and C and between X and S. Therefore, both parent subnet TAs C and S run an identity exchange with child subnet TH X. Both have the same encrypted keys file and X the common parameters file.

\subsection*{9.3.4 Configuration - Authentication Schemes}

Autokey has an intimidating number of options, most of which are not necessary in typical scenarios. However, the Trusted Certificate (TC) scheme is recommended for national NTP time services, such as those operated by NIST and USNO. Configuration for TC is very simple.

Referring to Figure 9.3, for each TH, A, B, R and X, as root:

\begin{verbatim}
# cd /usr/local/etc
# ntp-keygen -T
\end{verbatim}

and for the other hosts the same commands without the \(-T\) option. This generates an RSA private/public host key file and a self-signed certificate file for the RSA digital signature algorithm with the MD5 message digest algorithm. For the THs a trusted certificate is generated; for the others a nontreusted certificate is generated. Include in the \texttt{ntp.conf} configuration file for all hosts other than the primary servers, A, B and R, something like:

\begin{verbatim}
server <host> autokey
crypto
driftfile /etc/ntp.drift
\end{verbatim}

where \(<\text{host}>\) is the selected server name as shown in the figure. Servers A, B and R are configured for local reference clocks or trusted remoter servers as required.

In the above configuration examples, the default host name is the string returned by the Unix \texttt{gethostname()} routine, ordinarily the DNS name of the host. This name is used as the subject and issuer names on the certificate, as well as the default password for the encrypted keys file. The host name can be changed using the \(-s\) option of the \texttt{ntp-keygen} program. The default password can be changed using the \(-p\) option of the \texttt{ntp-keygen} program and the \texttt{pw} option of the \texttt{crypto} configuration command.

Group names can be added to this configuration by including the \(-s <\text{host}>@<\text{group}>\) option with the \texttt{ntp-keygen} program. For the purpose of illustration, the \(<\text{host}>\) string is empty, signifying the default host name. For example, \(<\text{yellow}>\) can be used for the Alice group, \(<\text{orange}>\) for the Helen group and \(<\text{blue}>\) for the
Carol group. In addition, for TH X the `ident yellow` option should be added to the `server` command for the Alice group and the `ident orange` option should be added to the `server` command for the Helen group.

### 9.3.5 Configuration - Identity Schemes

The example in this section uses the IFF identity scheme, but others, including GQ and MV, can be used as well. It's best to start with a functioning TC configuration and add commands as necessary. We start with the subnets of Figure 9.3 configured as in the previous section. Recall that the parent subnet TA for Alice is C and for Helen is S. Each of the TAs generates an encrypted server keys file and nonencrypted client parameters file for the IFF identity scheme using the `-I` option of the `ntp-keygen` program. Note the TAs are not necessarily trusted hosts, so may not need the `-T` option.

The nonencrypted client parameters can be exported using the command:

```bash
ntp-keygen -e >file
```

where the `-e` option redirects the client parameters to `file` via the standard output stream for a mail application or stored locally for later distribution to one or more THs. In a similar fashion the encrypted keys file can be exported using the command:

```bash
ntp-keygen -q <passwd2> >file
```

where `<passwd2>` is the read password for another TA. We won't need this file here.

While the file names used for the exported files are arbitrary, it is common practice to use the name given as the first line in the file with the filestamp suppressed. Thus, the nonencrypted parameters file from each TA is copied to X with this name.

To complete the configuration, the TH includes the client parameters file name in the `ident` option of the `server` command for the TA association

```bash
server 1.2.3.4 ident <group>,
```

where `<group>` is the file name given above.

### 9.3.6 Identity Schemes and Cryptotypes

A specific combination of authentication and identity schemes is called a cryptotype, which applies to clients and servers separately. A group can be configured using more than one cryptotype combination, although not all combinations are interoperable. Note however that some cryptotype combinations may successfully intertemperate with each other, but may not represent good security practice. The server and client cryptotypes are defined by the the following codes.

- **NONE** A client or server is type NONE if authentication is not available or not configured. Packets exchanged between client and server have no MAC.

- **AUTH** A client or server is type AUTH if the `key` option is specified with the `server` configuration command and the client and server keys are compatible. Packets exchanged between clients and servers have a MAC.

- **PC** A client or server is type PC if the `autokey` option is specified with the `server` configuration command and compatible host key and private certificate files are present. Packets exchanged between clients and servers have a MAC.

- **TC** A client or server is type TC if the `autokey` option is specified with the `server` configuration command and compatible host key and public certificate files are present. Packets exchanged between clients and servers have a MAC.
IDENT A client or server is type IDENT if the autokey option is specified with the server configuration command and compatible host key, public certificate and identity scheme files are present. Packets exchanged between clients and servers have a MAC.

The compatible cryptotypes for clients and servers are listed in the following table.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>AUTH</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>no</td>
<td>yes*</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

* These combinations are not valid if the restriction list includes the notrust option.

9.3.7 Error Codes

Errors can occur due to mismatched configurations, unexpected protocol restarts, expired certificates and unfriendly people. In most cases the protocol state machine recovers automatically by retransmission, timeout and restart, where necessary. Some errors are due to mismatched keys, digest schemes or identity schemes and must be corrected by installing the correct media and/or correcting the configuration file. One of the most common errors is expired certificates, which must be regenerated and signed at least once per year using the ntp-keygen program.

The following error codes are reported via the NTP control and monitoring protocol trap mechanism and to the cryptostats monitoring file if configured.

101 bad field format or length The packet has invalid version, length or format.
102 bad timestamp The packet timestamp is the same or older than the most recent received. This could be due to a replay or a server clock time step.
103 bad filestamp The packet filestamp is the same or older than the most recent received. This could be due to a replay or a key file generation error.
104 bad or missing public key The public key is missing, has incorrect format or is an unsupported type.
105 unsupported digest type The server requires an unsupported digest/signature scheme.
106 unsupported identity type The client or server has requested an identity scheme the other does not support.
107 bad signature length The signature length does not match the current public key.
108 signature not verified The message fails the signature check. It could be bogus or signed by a different private key.
109 certificate not verified The certificate is invalid or signed with the wrong key.
110 host certificate expired The old server certificate has expired.
111 bad or missing cookie The cookie is missing, corrupted or bogus.
112 bad or missing leapseconds table The leapseconds table is missing, corrupted or bogus.
113 bad or missing certificate The certificate is missing, corrupted or bogus.
114 bad or missing group key The identity key is missing, corrupt or bogus.
115 protocol error The protocol state machine has wedged due to unexpected restart.
9.3.8 Files

See the ntp-keygen page. Note that provisions to load leap second values from the NIST files have been removed. These provisions are now available whether or not the OpenSSL library is available. However, the functions that can download these values from servers remains available.

9.4 Orphan Mode

Sometimes an NTP subnet becomes isolated from all UTC sources such as local reference clocks or Internet time servers. In such cases it may be necessary that the subnet servers and clients remain synchronized to a common timescale, not necessarily the UTC timescale. Previously, this function was provided by the local clock driver to simulate a UTC source. A server with this driver could be used to synchronize other hosts in the subnet directly or indirectly.

There are many disadvantages using the local clock driver, primarily that the subnet is vulnerable to single-point failures and multiple server redundancy is not possible. Orphan mode is intended to replace the local clock driver. It provides a single simulated UTC source with multiple servers and provides seamless switching as servers fail and recover.

A common configuration for private networks includes one or more core servers operating at the lowest stratum. Good practice is to configure each of these servers as backup for the others using symmetric or broadcast modes. As long as at least one core server can reach a UTC source, the entire subnet can synchronize to it.

If no UTC sources are available to any core server, one of them can provide a simulated UTC source for all other hosts in the subnet. However, only one core server can simulate the UTC source and all direct dependents, called orphan children, must select the same server, called the orphan parent.

Hosts sharing the same common subnet, including potential orphan parents and potential orphan children, can be enabled for orphan mode using the \texttt{orphan stratum} option of the \texttt{tos} command, where \texttt{stratum} is some stratum less than 16 and greater than any anticipated stratum that might occur with configured Internet time servers. However, sufficient headroom should remain so every subnet host dependent on the orphan children has stratum less than 16. Where no associations for other servers or reference clocks are configured, the orphan stratum can be set to 1. These are the same considerations that guide the local clock driver stratum selection.

In order to avoid premature enabling orphan mode, a holdoff delay occurs when the daemon is first started and when all sources have been lost after that. The delay is intended to allow time for other sources to become reachable and selectable. Only when the delay has expired with no sources will orphan mode be enabled. By default the delay is 300 s (five minutes), but this can be changed using the \texttt{orphanwait} option of the \texttt{tos} command.

A orphan parent with no sources shows reference ID LOOP if operating at stratum 1 and 127.0.0.1 (IPv4 loopback address) otherwise. While ordinary NTP clients use a selection metric based on delay and dispersion, orphan children use a metric computed from the IP address of each core server. Each orphan child chooses the orphan parent as the core server with the smallest metric.

For orphan mode to work well, each core server with available sources should operate at the same stratum. All core servers and orphan children should include the same tos command in the configuration file. Each orphan child should include in the configuration file all root servers.

For example, consider the peer network configuration in Figure 9.5, where two or more campus primary or secondary (stratum 2) servers are configured with reference clocks or public Internet primary servers and with each other using symmetric modes. With this configuration a server that loses all sources continues to discipline the system clock using the other servers as backup. Only the core servers and orphan children need to be enabled for orphan mode.

For broadcast networks each core server is configured in both broadcast server and broadcast client modes as shown in Figure 9.6. Orphan children operate as broadcast clients of all core servers. As in peer networks, the core servers back up each other and only they and the orphan children need to be enabled for orphan mode.
Figure 9.5: Orphan Peer Configuration

Figure 9.6: Orphan Broadcast Configuration
In normal operation subnet hosts operate below stratum 5, so the subnet is automatically configured as described in the NTP specification. If all UTC sources are lost, all core servers become orphans and the orphan children will select the same core server to become the orphan parent.

9.5 NTP Interleaved Modes

In the protocol described in the NTP specification and reference implementation up to now, the transmit timestamp, which is captured before the message digest is computed and the packet queued for output, is properly called as a softstamp. The receive timestamp, which is captured after the input driver interrupt routine and before the packet is queued for input, is properly called a drivestamp. For enhanced accuracy it is desirable to capture the transmit timestamp as close to the wire as possible; for example, after the output driver interrupt routine.

In other words, we would like to replace the transmit softstamp with a drivestamp, but the problem is the transmit drivestamp is available only after the packet has been sent. A solution for this problem is the two-step or interleaved protocol described on this page and included in the current reference implementation. In interleaved modes the transmit drivestamp for one packet is actually carried in the immediately following packet. The trick, however, is to implement the interleaved protocol without changing the NTP packet header format, without compromising backwards compatibility and without compromising the error recovery properties.

The reference implementation captures a softstamp before the message digest routine and a drivestamp after the output interrupt routine. In this design the latter timestamp can be considered most accurate, as it avoids the various queuing and transmission latencies. The difference between the two timestamps, which is called the interleaved or output delay, varies from 16 $\mu$s for a dual-core Pentium running FreeBSD 6.1 to 1100 $\mu$s for a Sun Blade 1500 running Solaris 10.

Interleaved mode can be used only in NTP symmetric and broadcast modes. It is activated by the \texttt{xleave} option with the \texttt{peer} or \texttt{broadcast} configuration commands. A broadcast server configured for interleaved mode is transparent to ordinary broadcast clients, so both ordinary and interleaved broadcast clients can use the same packets. An interleaved symmetric active peer automatically switches to ordinary symmetric mode if the other peer is not capable of operation in interleaved mode.

As demonstrated in the white paper Analysis and Simulation of the NTP On-Wire Protocols, the interleaved modes have the same resistance to lost packets, duplicate packets, packets crossed in flight and protocol restarts as the ordinary modes. An application of the interleaved symmetric mode in space missions is presented in the white paper Time Synchronization for Space Data Links.

9.6 The Huff-n'-Puff Filter

In scenarios where a considerable amount of data are downloaded or uploaded using DSL or telephone modem lines, timekeeping quality can be seriously degraded. This occurs because the traffic volume, and thus the queuing delays, on the upload and download directions of transmission can be very different. In many cases the apparent time errors are so large as to exceed the step threshold and a step correction can occur during and after the data transfer.

The huff-n’-puff filter is designed to correct the apparent time offset in these cases. It depends on knowledge of the propagation delay when no other traffic is present, such as during other than work hours. The filter remembers the minimum delay over the most recent interval measured usually in hours. Under conditions of large delay, the filter corrects the apparent offset using the sign of the offset and the difference between the apparent delay and minimum delay. The name of the filter reflects the negative (huff) and positive (puff) correction, which depends on the sign of the offset. The filter is activated by the \texttt{tinker huffpuff} command, as described in the Miscellaneous Options page.

Figure 9.7 shows how the huff-n’-puff filter works. Recall from the Clock Filter Algorithm page that the wedge scattergram plots sample points $(x, y)$ corresponding to the measured delay and offset, and that the limb lines are at slope $\pm0.5$. Note in the figure that the samples are clustered close to the upper limb line, representing heavy traffic in...
the download direction. The apparent offset $y_0$ is near zero at the minimum delay $x_0$, which is near 0.1s. Thus, for a point $(x, y)$, the true offset is

$$\theta = y - (x - x_0) / 2$$ for $y > y_0$ at or near the upper limb line or

$$\theta = y + (x - x_0) / 2$$ for $y < y_0$ at or near the lower limb line.

In either case the associated delay is $\delta = x$.

In the interior of the wedge scattergram far from the limb lines, the corrections are less effective and can lead to significant errors if the area between the limb lines is heavily populated.

### 9.7 Clock Filter Algorithm

The clock filter algorithm processes the offset and delay samples produced by the on-wire protocol for each peer process separately. It uses a sliding window of eight samples and picks out the sample with the least expected error. This page describes the algorithm design principles along with an example of typical performance.

Figure 9.8 shows a typical wedge scattergram plotting sample points of offset versus delay collected over a 24-hr period. As the delay increases, the offset variation increases, so the best samples are those at the lowest delay. There are two limb lines at slope $\pm 0.5$, representing the limits of sample variation. However, it is apparent that, if a way could be found to find the sample of lowest delay, it would have the least offset variation and would be the best candidate to synchronize the system clock.

The clock filter algorithm works best when the delays are statistically identical in the reciprocal directions between the server and client. This is apparent in Figure 9.8, where the scattergram is symmetric about the x axis through the apex sample. In configurations where the delays are not reciprocal, or where the transmission delays on the two directions are traffic dependent, this may not be the case. A common case with DSL links is when downloading or uploading a large file. During the download or upload process, the delays may be significantly different resulting in large errors. However, these errors can be largely eliminated using samples near the limb lines, as described on the The Huff-n’-Puff Filter page.

In the clock filter algorithm the offset and delay samples from the on-wire protocol are inserted as the youngest stage
Figure 9.8: Wedge Scattergram
of an eight-stage shift register, thus discarding the oldest stage. Each time an NTP packet is received from a source, a dispersion sample is initialized as the sum of the precisions of the server and client. Precision is defined by the latency to read the system clock and varies from 1000 ns to 100 ns in modern machines. The dispersion sample is inserted in the shift register along with the associated offset and delay samples. Subsequently, the dispersion sample in each stage is increased at a fixed rate of $15 \mu s/s$, representing the worst case error due to skew between the server and client clock frequencies.

In each peer process the clock filter algorithm selects the stage with the smallest delay, which generally represents the most accurate data, and it and the associated offset sample become the peer variables of the same name. The peer jitter statistic is computed as the root mean square (RMS) differences between the offset samples and the offset of the selected stage.

The peer dispersion statistic is determined as a weighted sum of the dispersion samples in the shift register. Initially, the dispersion of all shift register stages is set to a large number “infinity” equal to 16 s. The weight factor for each stage, starting from the youngest numbered $i = 1$, is $2^{-i}$, which means the peer dispersion is approximately 16 s.

As samples enter the register, the peer dispersion drops from 16 s to 8 s, 4 s, 2 s, and so forth. In practice, the synchronization distance, which is equal to one-half the delay plus the dispersion, falls below the select threshold of 1.5 s in about four updates. This gives some time for meaningful comparison between sources, if more than one are available. The dispersion continues to grow at the same rate as the sample dispersion. For additional information on statistical principles and performance metrics, see the Performance Metrics page.

As explained elsewhere, when a source becomes unreachable, the poll process inserts a dummy infinity sample in the shift register for each poll sent. After eight polls, the register returns to its original state.

---

**Figure 9.9: Raw Offsets**

*Figure 9.9 and Figure 9.10 show the performance of the algorithm for a typical Internet path over a 24-hr period. Figure 9.9 shows the raw offsets produced by the on-wired protocol, while Figure 9.10 shows the filtered offsets produced by the clock filter algorithm. If we consider the series formed as the absolute value of the offset samples, the mean error is defined as the mean of this series. Thus, the mean error of the raw samples is 0.724 ms, while the mean error of the filtered series is 0.192 ms. Radio engineers would interpret this as a processing gain of 11.5 dB.*

The reader might notice the somewhat boxy characteristic of the filtered offsets. Once a sample is selected, it remains selected until a newer sample with lower delay is available. This commonly occurs when an older selected sample is discarded from the shift register. The reason for this is to preserve causality; that is, time always moves forward, never backward. The result can be the loss of up to seven samples in the shift register, or more to the point, the output sample rate can never be less than one in eight input samples. The clock discipline algorithm is specifically designed to operate at this rate.
9.8 Clock Select Algorithm

The clock select algorithm determines from a set of sources, which are correct (truechimers) and which are not (falsetickers) according to a set of formal correctness assertions. The principles are based on the observation that the maximum error in determining the offset of a candidate cannot exceed one-half the roundtrip delay to the primary reference clock at the time of measurement. This must be increased by the maximum error that can accumulate since then. The selection metric, called the root distance, is one-half the roundtrip root delay plus the root dispersion plus minor error contributions not considered here.

First, a number of sanity checks is performed to sift the selectable candidate from among the source population. The sanity checks are summarized as follows:

1. A stratum error occurs if (1) the source had never been synchronized or (2) the stratum of the source is below the floor option or not below the ceiling option of the tos command. The default values for these options are 0 and 15, respectively. Note that 15 is a valid stratum, but a server operating at that stratum cannot synchronize clients.

2. A distance error occurs for a source if the root distance (also known as synchronization distance) of the source is not below the distance threshold maxdist option of the tos command. The default value for this option is 1.5 s for networks including only the Earth, but this should be increased to 2.5 s for networks including the Moon.

3. A loop error occurs if the source is synchronized to the client. This can occur if two peers are configured with each other in symmetric modes.

4. An unreachable error occurs if the source is unreachable or if the server or peer command for the source includes the noselect option.

Sources showing one or more of these errors are considered nonselectable; only the selectable candidates are considered in the following algorithm. Given the measured offset $\theta_0$ and root distance $\lambda$, this defines a correctness interval $[\theta_0 - \lambda, \theta_0 + \lambda]$ of points where the true value of $\theta$ lies somewhere on the interval. The given problem is to determine from a set of correctness intervals, which represent truechimers and which represent falsetickers. The principles must be given a precise definition. The intersection interval is the smallest interval containing points from the largest number of correctness intervals. An algorithm that finds the intersection interval was devised by Keith Marzullo in his doctoral dissertation. It was first implemented in the Digital Time Synchronization Service (DTSS) for the VMS operating system for the VAX.

While the NTP algorithm is based on DTSS, it remains to establish which point in the correctness interval represents the best estimate of the offset for each candidate. The best point is at the midpoint $\theta_0$ of the correctness interval; however, the midpoint might not be within the intersection interval. A candidate with a correctness interval that
contains points in the intersection interval is a truechimer and the best offset estimate is the midpoint of its correctness interval. A candidate with a correctness interval that contains no points in the intersection interval is a falseticker.

Figure 9.11 shows correctness intervals for each of four candidates A, B, C and D. We need to find the maximum number of candidates that contain points in common. The result is the interval labeled DTSS. In the figure there are three truechimers A, B and C, and one falseticker D. In DTSS any point in the intersection interval can represent the true time; however, as shown below, this may throw away valuable statistical information. In any case, the clock is considered correct if the number of truechimers found in this way are greater than half the total number of candidates.

The question remains, which is the best point to represent the true time of each correctness interval? Fortunately, we already have the maximum likelihood estimate at the midpoint of each correctness interval. But, while the midpoint of candidate C is outside the intersection interval, its correctness interval contains points in common with the intersection interval, so the candidate is a truechimer and the midpoint is chosen to represent its time.

The DTSS correctness assertions do not consider how best to represent the truechimer time. To support the midpoint choice, consider the selection algorithm as a method to reject correctness intervals that cannot contribute to the final outcome; that is, they are falsetickers. The remaining correctness intervals can contribute to the final outcome; that is, they are truechimers. Samples in the intersection interval are usually of very low probability and thus poor estimates for truechimer time. On the other hand, the midpoint sample produced by the clock filter algorithm is the maximum likelihood estimate and thus best represents the truechimer time.

The algorithm operates as shown in Figure 9.12. Let $m$ be the number of candidates and $f$ the number of falsetickers, initially zero. Move a pointer from the leftmost endpoint towards the rightmost endpoint in Figure 9.11 and count the number of candidates, stopping when that number reaches $m - f$; this is the left endpoint of the intersection interval. Then, do the same, but moving from the rightmost endpoint towards the leftmost endpoint; this is the right endpoint of the intersection interval. If the left endpoint is less than the right endpoint, the intersection interval has been found. Otherwise, increase $f$ by 1. If $f$ is less than $n / 2$, try again; otherwise, the algorithm fails and no truechimers could be found.

The clock select algorithm again scans the correctness intervals. If the right endpoint of the correctness interval for a candidate is greater than the left endpoint of the intersection interval, or if the left endpoint of the correctness interval is less than the right endpoint of the intersection interval, the candidate is a truechimer; otherwise, it is a falseticker.

In practice, with fast LANs and modern computers, the correctness interval can be quite small, especially when the candidates are multiple reference clocks. In such cases the intersection interval might be empty, due to insignificant differences in the reference clock offsets. To avoid this, the size of the correctness interval is padded to the value of mindist, with default 1 ms. This value can be changed using the mindist option of the tos command.

### 9.9 Clock Cluster Algorithm

The clock cluster algorithm processes the truechimers produced by the clock select algorithm to produce a list of survivors. These survivors are used by the mitigation algorithms to discipline the system clock. The cluster algorithm operates in a series of rounds, where at each round the truechimer furthest from the offset centroid is pruned from the
First, the truechimer associations are saved on an unordered list with each candidate entry identified with index \( i \) \((i = 1, \ldots, n)\), where \( n \) is the number of candidates. Let \( \theta(i) \), be the offset and \( \lambda(i) \) be the root distance of the \( i \)th entry. Recall that the root distance is equal to the root dispersion plus half the root delay. For the \( i \)th candidate on the list, a statistic called the select jitter relative to the \( i \)th candidate is calculated as follows. Let

\[
d_i(j) = |\theta(j) - \theta(i)| \lambda(i),
\]

where \( \theta(i) \) is the peer offset of the \( i \)th entry and \( \theta(j) \) is the peer offset of the \( j \)th entry, both produced by the clock filter algorithm. The metric used by the cluster algorithm is the select jitter \( \phi_S(i) \) computed as the root mean square (RMS) of the \( d_i(j) \) as \( j \) ranges from 1 to \( n \). For the purpose of notation in the example to follow, let \( \phi_R(i) \) be the peer jitter computed by the clock filter algorithm for the \( i \)th candidate.

The object at each round is to prune the entry with the largest metric until the termination condition is met. Note that the select jitter must be recomputed at each round, but the peer jitter does not change. At each round the remaining entries on the list represent the survivors of that round. If the candidate to be pruned is preemptable and the number of candidates is greater than the maxclock threshold, the association is demobilized. This is useful in the schemes described on the Automatic Server Discovery Schemes page. The maxclock threshold default is 10, but it can be changed using the maxclock option of the tos command. Further pruning is subject to the following termination conditions, but no associations will be automatically demobilized.

The termination condition has two parts. First, if the number of survivors is not greater than the minclock threshold set by the minclock option of the tos command, the pruning process terminates. The minclock default is 3, but can be changed to fit special conditions, as described on the Mitigation Rules and the prefer Keyword page.

The second termination condition is more intricate. Figure 9.13 shows a round where a candidate of (a) is pruned to yield the candidates of (b). Let \( \phi_{max} \) be the maximum select jitter and \( \phi_{min} \) be the minimum peer jitter over all candidates on the list. In (a), candidate 1 has the highest select jitter, so \( \phi_{max} = \phi_S(1) \). Candidate 4 has the lowest peer jitter, so \( \phi_{min} = \phi_R(4) \). Since \( \phi_{max} > \phi_{min} \), select jitter dominates peer jitter, the algorithm prunes candidate 1. In (b), \( \phi_{max} = \phi_S(3) \) and \( \phi_{min} = \phi_R(4) \). Since \( \phi_{max} < \phi_{min} \), pruning additional candidates does not reduce select jitter, the
algorithm terminates with candidates 2, 3 and 4 as survivors.

The survivor list is passed on to the mitigation algorithms, which combine the survivors, select a system peer, and compute the system statistics passed on to dependent clients. Note the use of root distance $\lambda$ as a weight factor at each round in the clock cluster algorithm. This is to favor the survivors with the lowest root distance and thus the smallest maximum error.

### 9.10 Mitigation Rules and the prefer Keyword

This page summarizes the criteria for choosing from among the survivors of the clock cluster algorithm a set of contributors to the clock discipline algorithm. The criteria are very meticulous, since they have to handle many different scenarios that may be optimized for special circumstances, including some scenarios designed to support planetary and deep space missions. For additional information on statistical principles and performance metrics, see the Performance Metrics page.

Recall the suite of NTP data acquisition and grooming algorithms. These algorithms proceed in five phases. Phase one discovers the available sources and mobilizes an association for each source found. These sources can result from explicit configuration, broadcast discovery or the pool and manycast autonomous configuration schemes. See the Automatic Server Discovery Schemes page for further information.

Phase two selects the candidates from among the sources by excluding those sources showing one or more of the errors summarized on the Clock Select Algorithm page and to determine the truechimers from among the candidates, leaving behind the falsetickers. A server or peer configured with the `true` option is declared a truechimer independent of this algorithm. Phase four uses the algorithm described on the Clock Cluster Algorithm page to prune the statistical outliers from the truechimers, leaving the survivor list as result.

Phase five uses a set of algorithms and mitigation rules to combined the survivor statistics and discipline the system clock. The mitigation rules select from among the survivors a system peer from which a set of system statistics can be inherited and passed along to dependent clients, if any. The mitigation algorithms and rules are the main topic of this page. The clock offset developed from these algorithms can discipline the system clock, either using the clock discipline algorithm or using the kernel to discipline the system clock directly, as described on the A Kernel Model for Precision Timekeeping page.
9.10.1 Combine Algorithm

The clock combine algorithm uses the survivor list to produce a weighted average of both offset and jitter. Absent other considerations discussed later, the combined offset is used to discipline the system clock, while the combined jitter is augmented with other components to produce the system jitter statistic inherited by dependent clients, if any.

The clock combine algorithm uses a weight factor for each survivor equal to the reciprocal of the root distance. This is normalized so that the sum of the reciprocals is equal to unity. This design favors the survivors at the smallest root distance and thus the smallest maximum error.

9.10.2 Anti-Clockhop Algorithm

The anti-clockhop algorithm is intended for cases where multiple servers are available on a fast LAN with modern computers. Typical offset differences between servers in such cases are less than 0.5 ms. However, changes between servers can result in unnecessary system jitter. The object of the anti-clockhop algorithm is to avoid changing the current system peer, unless it becomes stale or has significant offset relative to other candidates on the survivor list.

For the purposes of the following description, call the last selected system peer the old peer, and the currently selected source the candidate peer. At each update, the candidate peer is selected as the first peer on the survivor list sorted by increasing root distance. The algorithm initializes the clockhop threshold with the value of mindist, by default 1 ms.

The anti-clockhop algorithm is called immediately after the combine algorithm. If there was no old peer or the old and candidate peers are the same, the candidate peer becomes the system peer. If the old peer and the candidate peer are different, the algorithm measures the difference between the offset of the old peer and the candidate peer. If the difference exceeds the clockhop threshold, the candidate peer becomes the system peer and the clockhop threshold is restored to its original value. If the difference is less than the clockhop threshold, the old peer continues as the system peer. However, at each subsequent update, the algorithm reduces the clockhop threshold by half. Should operation continue in this way, the candidate peer will eventually become the system peer.

9.10.3 Peer Classification

The behavior of the various algorithms and mitigation rules involved depends on how the various synchronization sources are classified. This depends on whether the source is local or remote and if local, the type of source. The following classes are defined:

1. A selectable association configured for a remote server or peer is classified as a client association. All other selectable associations are classified as device driver associations of one kind or another. In general, one or more sources of either type will be available in each installation.

2. If all sources have been lost and one or more hosts on a common DMZ network have specified the orphan stratum in the orphan option of the tos command, each of them can become an orphan parent. Dependent orphan children on the same DMZ network will see the orphan parents as if synchronized to a server at the orphan stratum. Note that, as described on the Orphan Mode page, all orphan children will select the same orphan parent for synchronization.

3. When a device driver has been configured for pulse-per-second (PPS) signals and PPS signals are being received, it is designated the PPS driver. Note that the Pulse-per-Second driver (type 22) is often used as a PPS driver, but any driver can be configured as a PPS driver if the hardware facilities are available. The PPS driver provides precision clock discipline only within ±0.4 s, so it is always associated with another source or sources that provide the seconds numbering function.

4. When the Undisciplined Local Clock driver (type 1) is configured, it is designated the local driver. It can be used either as a backup source (stratum greater than zero) should all sources fail, or as the primary source (stratum zero) whether or not other sources are available if the prefer option is present. The local driver can be used when the kernel time is disciplined by some other means of synchronization, such as the NIST lock clock.
scheme, or another synchronization protocol such as the IEEE 1588 Precision Time Protocol (PTP) or Digital Time Synchronization Service (DTSS).

5. When the Automated Computer Time Service driver (type 18) is configured, it is designated the *modem driver*. It is used either as a backup source, should all other sources fail, or as the primary source if the *prefer* option is present.

### 9.10.4 The prefer Peer

The mitigation rules are designed to provide an intelligent selection of the system peer from among the selectable sources of different types. When used with the `server` or `peer` commands, the *prefer* option designates one or more sources as preferred over all others. While the rules do not forbid it, it is usually not useful to designate more than one source as preferred; however, if more than one source is so designated, they are used in the order specified in the configuration file. If the first one becomes un selectable, the second one is considered and so forth. This order of priority is also applicable to multiple PPS drivers, multiple modem drivers and even multiple local drivers, although that would not normally be useful.

The cluster algorithm works on the set of truechimers produced by the select algorithm. At each round the algorithm casts off the survivor least likely to influence the choice of system peer. If selectable, the prefer peer is never discarded; on the contrary, its potential removal becomes a termination condition. However, the prefer peer can still be discarded by the select algorithm as a falseticker; otherwise, the prefer peer becomes the system peer.

Ordinarily, the combine algorithm computes a weighted average of the survivor offset and jitter to produce the final values. However, if a prefer peer is among the survivors, the combine algorithm is not used. Instead, the offset and jitter of the prefer peer are used exclusively as the final values. In the common case involving a radio clock and a flock of remote backup servers, and with the radio clock designated a prefer peer, the radio clock disciplines the system clock as long as the radio itself remains operational. However, if the radio fails or becomes a falseticker, the combined backup sources continue to discipline the system clock.

### 9.10.5 Mitigation Rules

As the select algorithm scans the associations for selectable candidates, the modem driver and local driver are segregated for later, but only if not designated a prefer peer. If so designated, the driver is included among the candidate population. In addition, if orphan parents are found, the parent with the lowest metric is segregated for later; the others are discarded. For this purpose the metric is defined as the four-octet IPv4 address or the first four octets of the hashed IPv6 address. The resulting candidates, including any prefer peers found, are processed by the select algorithm to produce a possibly empty set of truechimers.

As previously noted, the cluster algorithm casts out outliers, leaving the survivor list for later processing. The survivor list is then sorted by increasing root distance and the first entry temporarily designated the system peer. At this point the following contributors to the system clock discipline may be available:

- (potential) system peer, if there are survivors;
- orphan parent, if present;
- local driver, if present;
- modem driver, if present;
- prefer peer, if present;
- PPS driver, if present.

The mitigation algorithm proceeds in three steps in turn.

1. If there are no survivors, the modem driver becomes the only survivor if there is one. If not, the local driver becomes the only survivor if there is one. If not, the orphan parent becomes the only survivor if there is one.
If the number of survivors at this point is less than the `minsane` option of the `tos` command, the algorithm is terminated and the system variables remain unchanged. Note that `minsane` is by default 1, but can be set at any value including 0.

2. If the prefer peer is among the survivors, it becomes the system peer and its offset and jitter are inherited by the corresponding system variables. Otherwise, the combine algorithm computes these variables from the survivor population.

3. If there is a PPS driver and the system clock offset at this point is less than 0.4 s, and if there is a prefer peer among the survivors or if the PPS peer is designated as a prefer peer, the PPS driver becomes the system peer and its offset and jitter are inherited by the system variables, thus overriding any variables already computed. Note that a PPS driver is present only if PPS signals are actually being received and enabled by the associated driver.

If none of the above is the case, the data are disregarded and the system variables remain as they are.

### 9.10.6 The `minsane` Option

The `minsane` option of the `tos` command, the `prefer` option of the `server` and `peer` commands and the `flag` option of the `fudge` command for a selected driver can be used with the mitigation rules to provide many useful configurations. The `minsane` option specifies the minimum number of survivors required to synchronize the system clock. The `prefer` option operates as described in previous sections. The `flag` option enables the PPS signal for the selected driver.

A common scenario is a GPS driver with a serial timecode and PPS signal. The PPS signal is disabled until the system clock has been set by some means, not necessarily the GPS driver. If the serial timecode is within 0.4 s of the PPS signal, the GPS driver is designated the PPS driver and the PPS signal disciplines the system clock. If the serial timecode becomes unreliable, or if the PPS signal is disconnected, the GPS driver stops updating the system clock and so eventually becomes unreachable and is replaced by other sources.

Whether or not the GPS driver disables the PPS signal when the timecode becomes unreliable is at the discretion of the driver. Ordinarily, the PPS signal is disabled in this case; however, when the GPS receiver has a precision holdover oscillator, the driver may elect to continue PPS discipline. In this case, `minsane` can be set to zero so the PPS signal continues to discipline the system clock.

### 9.11 Clock Discipline Algorithm

#### 9.11.1 General Overview

At the heart of the NTP specification and reference implementation is the clock discipline algorithm, which is best described as an adaptive parameter, hybrid phase/frequency-lock feedback loop. It is an intricately crafted algorithm that automatically adapts for optimum performance while minimizing network overhead. Operation is in two modes, phase-lock loop (PLL), which is used at poll intervals below the Allan intercept, by default 2048 s, and frequency-lock loop (FLL), which is used above that.

#### 9.11.2 Clock Discipline Operations

A block diagram of the clock discipline is shown in Figure 9.14. The timestamp of a reference clock or remote server is compared with the timestamp of the system clock, represented as a variable frequency oscillator (VFO), to produce a raw offset sample $V_d$. Offset samples are processed by the clock filter to produce a filtered update $V_s$. The loop filter implements a type-2 proportional-integrator controller (PIC). The PIC can minimize errors in both time and frequency using predictors $x$ and $y$, respectively. The clock adjust process samples these predictors once each second for the daemon discipline or once each tick interrupt for the kernel discipline to produce the system clock update $V_c$.
In PLL mode the frequency predictor is an integral of the offset over past updates, while the phase predictor is the offset amortized over time in order to avoid setting the clock backward. In FLL mode the phase predictor is not used, while the frequency predictor is similar to the NIST lockclock algorithm. In this algorithm, the frequency predictor is computed as a fraction of the current offset divided by the time since the last update in order to minimize the offset at the next update.

The discipline response in PLL mode is determined by the time constant, which results in a “stiffness” depending on the jitter of the available sources and the wander of the system clock oscillator. The scaled time constant is also used as the poll interval described on the Poll Process page. However, in NTP symmetric mode, each peer manages its own poll interval and the two might not be the same. In such cases either peer uses the minimum of its own poll interval and that of the other peer, which is included in the NTP packet header.

### 9.11.3 Loop Dynamics

It is necessary to verify that the clock discipline algorithm is stable and satisfies the Nyquist criterion, which requires that the sampling rate be at least twice the bandwidth. In this case the bandwidth can be approximated by the reciprocal of the time constant. In the NTP specification and reference implementation, time constants and poll intervals are expressed as exponents of 2. By construction, the time constant exponent is five times the poll interval exponent. Thus, the default poll exponent of 6 corresponds to a poll interval of 64 s and a time constant of 2048 s. A change in the poll interval changes the time constant by a corresponding amount. The Nyquist criterion requires the sample interval to be not more than half the time constant or 1024 s. The clock filter guarantees at least one sample in eight poll intervals, so the sample interval is not more than 512 s. This would be described as oversampling by a factor of two. Finally, the PLL parameters have been chosen for a damping factor of 2, which results in a much faster risetime than with critical damping, but results in modest overshoot of 6 percent.

It is important to understand how the dynamics of the PLL are affected by the time constant and poll interval. At the default poll interval of 64 s and a step offset change of 100 ms, the time response crosses zero in about 50 min and overshoots about 6 ms, as per design. Ordinarily, a step correction would cause a temporary frequency surge of about 5 PPM, which along with the overshoot slowly dissipates over a few hours.

However, the clock state machine used with the discipline algorithm avoids this transient at startup. It does this using a previously saved frequency file, if present, or by measuring the oscillator frequency, if not. It then quickly amortizes the residual offset at startup without affecting the oscillator frequency. In this way the offset error is less than 0.5 ms within 5 min, if the file is present, and within 10 min if not. See the Clock State Machine page for further details.

Since the PLL is linear, the response with different offset step amplitudes and poll intervals has the same characteristic shape, but scaled differently in amplitude and time. The response scales exactly with step amplitude, so that the response to a 10-ms step has the same shape as at 64 s, but with amplitude compressed by one-tenth. The response
scales exactly with poll interval, so that response at a poll interval of 8 s has the same shape as at 64 s, but with time compressed by one-eighth.

The optimum time constant, and thus the poll interval, depends on the network time jitter and the oscillator frequency wander. Errors due to jitter decrease as the time constant increases, while errors due to wander decrease as the time constant decreases. For typical Internet paths, the two error characteristics intersect at a point called the *Allan intercept*, which represents the optimum time constant. With a compromise Allan intercept of 2048 s, the optimum poll interval is about 64 s, which corresponds to a compromise poll exponent of 6. For fast LANs with modern computers, the Allan intercept is somewhat lower at around 512 s, so a compromise poll exponent of 4 (16 s) is appropriate. An intricate, heuristic algorithm is used to manage the actual poll interval within a specified range. Details are on the Poll Process page.

In the NTPv4 specification and reference implementation a state machine is used to manage the system clock under exceptional conditions, as when the daemon is first started or when encountering severe network congestion. In extreme cases not likely to be encountered in normal operation, the system time can be stepped forward or backward more than 128 ms. Further details are on the Clock State Machine page.

### 9.11.4 Clock Initialization and Management

If left running continuously, an NTP client on a fast LAN in a home or office environment can maintain synchronization nominally within one millisecond. When the ambient temperature variations are less than a degree Celsius, the clock oscillator frequency is disciplined to within one part per million (PPM), even when the clock oscillator native frequency offset is 100 PPM or more.

For laptops and portable devices when the power is turned off, the battery backup clock offset error can increase as much as one second per day. When power is restored after several hours or days, the clock offset and oscillator frequency errors must be resolved by the clock discipline algorithm, but this can take several hours without specific provisions.

The provisions described in this section insure that, in all but pathological situations, the startup transient is suppressed to within nominal levels in no more than five minutes after a warm start or ten minutes after a cold start. Following is a summary of these provisions. A detailed discussion of these provisions is on the Clock State Machine page.

The reference implementation measures the clock oscillator frequency and updates a frequency file at intervals of one hour or more, depending on the measured frequency wander. This design is intended to minimize write cycles in NVRAM that might be used in a laptop or portable device. In a warm start, the frequency is initialized from this file, which avoids a possibly lengthy convergence time. In a cold start when no frequency file is available, the reference implementation first measures the oscillator frequency over a five-min interval. This generally results in a residual frequency error less than 1 PPM. The measurement interval can be changed using the stepout option of the tinker command.

In order to reduce the clock offset error at restart, the reference implementation mext disables oscillator frequency discipline and enables clock offset discipline with a small time constant. This is designed to quickly reduce the clock offset error without causing a frequency surge. This configuration is continued for an interval of five-min, after which the clock offset error is usually no more than a millisecond. The measurement interval can be changed using the stepout option of the tinker command.

Another concern at restart is the time necessary for the select and cluster algorithms to refine and validate the initial clock offset estimate. Normally, this takes several updates before setting the system clock. As the default minimum poll interval in most configurations is about one minute, it can take several minutes before setting the system clock. The iburst option of the server command changes the behavior at restart and is recommended for client/server configurations. When this option is enabled, the client sends a volley of six requests at intervals of two seconds. This usually insures a reliable estimate is available in about ten seconds before setting the clock. Once this initial volley is complete, the procedures described above are executed.

As a result of the above considerations, when a backup source, such as the local clock driver, ACTS modem driver or orphan mode is included in the system configuration, it may happen that one or more of them are selectable before one
or more of the regular sources are selectable. When backup sources are included in the configuration, the reference implementation waits an interval of several minutes without regular sources before switching to backup sources. This is generally enough to avoid startup transients due to premature switching to backup sources. The interval can be changed using the orphanwait option of the tos command.

### 9.12 Poll Process

The poll process sends NTP packets at intervals determined by the clock discipline algorithm. The process is designed to provide a sufficient update rate to maximize accuracy while minimizing network overhead. The process is designed to operate over a poll exponent range between 3 (8 s) and 17 (36 hr). The minimum and maximum poll exponent within this range can be set using the minpoll and maxpoll options of the server command, with default 6 (64 s) and 10 (1024 s), respectively.

The poll interval is managed by a heuristic algorithm developed over several years of experimentation. It depends on an exponentially weighted average of clock offset differences, called clock jitter, and a jiggle counter, which is initially set to zero. When a clock update is received and the offset exceeds the clock jitter by a factor of 4, the jiggle counter is increased by the poll exponent; otherwise, it is decreased by twice the poll exponent. If the jiggle counter is greater than an arbitrary threshold of 30, it is reset to 0 and the poll exponent is increased by 1. If the jiggle counter is less than -30, it is set to 0 and the poll exponent decreased by 1. In effect, the algorithm has a relatively slow reaction to good news, but a relatively fast reaction to bad news.

As an option of the server command, instead of a single packet, the poll process can send a burst of several packets at 2-s intervals. This is designed to reduce the time to synchronize the clock at initial startup (iburst) and/or to reduce the phase noise at the longer poll intervals (burst). The iburst option is effective only when the server is unreachable, while the burst option is effective only when the server is reachable. The two options are independent of each other and both can be enabled at the same time.

For the iburst option the number of packets in the burst is six, which is the number normally needed to synchronize the clock; for the burst option, the number of packets in the burst is determined by the difference between the current poll exponent and the minimum poll exponent as a power of 2. For instance, with the default minimum poll exponent of 6 (64 s), only one packet is sent for every poll, while the full number of eight packets is sent at poll exponents of 9 (512 s) or more. This insures that the average headway will never exceed the minimum headway.

The burst options can result in increased load on the network if not carefully designed. Both options are affected by the provisions described on the Rate Management and the Kiss-o’-Death Packet page. In addition, when iburst or burst are enabled, the first packet of the burst is sent, but the remaining packets sent only when the reply to the fist packet is received. If no reply has been received after a timeout set by the minpoll option, the first packet is sent again. This means that, even if a server is unreachable, the network load is no more than at the minimum poll interval.

To further reduce the network load when a server is unreachable, an unreach timer is incremented by 1 at each poll interval, but is set to 0 as each packet is received. If the timer exceeds the unreach threshold set at 10, the poll exponent is incremented by 1 and the unreach timer set to 0. This continues until the poll exponent reaches the maximum set by the maxpoll option.

### 9.13 Clock State Machine

In the NTPv4 specification and reference implementation a state machine is used to manage the system clock under exceptional conditions, as when the daemon is first started or when encountering severe network congestion. This page describes the design and operation of the state machine in detail.

The state machine is activated upon receipt of an update by the clock discipline algorithm. Its primary purpose is to determines whether the clock is slewed or stepped and how the initial time and frequency are determined using three thresholds: panic, step and stepout, and one timer: hold.
9.13.1 Panic Threshold

Most computers today incorporate a time-of-year (TOY) chip to maintain the time when the power is off. When the computer is restarted, the chip is used to initialize the operating system time. In case there is no TOY chip or the TOY time is different from NTP time by more than the panic threshold, the daemon assumes something must be terribly wrong, so exits with a message to the system operator to set the time manually. With the \texttt{-g} option on the command line, the daemon sets the clock to NTP time at the first update, but exits if the offset exceeds the panic threshold at subsequent updates. The panic threshold default is 1000 s, but it can be changed with the \texttt{panic} option of the \texttt{tinker} command.

9.13.2 Step and Stepout Thresholds

Under ordinary conditions, the clock discipline gradually slews the clock to the correct time, so that the time is effectively continuous and never stepped forward or backward. If, due to extreme network congestion, an offset spike exceeds the step threshold, by default 128 ms, the spike is discarded. However, if offset spikes greater than the step threshold persist for an interval more than the stepout threshold, by default 300 s, the system clock is stepped to the correct time.

In practice, the need for a step has been extremely rare and almost always the result of a hardware failure or operator error. The step threshold and stepout threshold can be changed using the \texttt{step} and \texttt{stepout} options of the \texttt{tinker} command, respectively. If the step threshold is set to zero, the step function is entirely disabled and the clock is always slewed. The daemon sets the step threshold to 600 s using the \texttt{-x} option on the command line. If the \texttt{-g} option is used or the step threshold is set greater than 0.5 s, the precision time kernel support is disabled.

Historically, the most important application of the step function was when a leap second was inserted in the Coordinated Universal Time (UTC) timescale and the kernel precision time support was not available. This also happened with older reference clocks that indicated an impending leap second, but the radio itself did not respond until it resynchronized some minutes later. Further details are on the \texttt{Leap Second Processing} page.

In some applications the clock can never be set backward, even if accidentally set forward a week by some evil means. The issues should be carefully considered before using these options. The slew rate is fixed at 500 parts-per-million (PPM) by the Unix kernel. As a result, the clock can take 33 minutes to amortize each second the clock is outside the acceptable range. During this interval the clock will not be consistent with any other network clock and the system cannot be used for distributed applications that require correctly synchronized network time.

9.13.3 Hold Timer

When the daemon is started after a considerable downtime, it could be the TOY chip clock has drifted significantly from NTP time. This can cause a transient at system startup. In the past, this has produced a phase transient and resulted in a frequency surge that could take some time, even hours, to subside. When the highest accuracy is required, some means is necessary to manage the startup process so that the the clock is quickly set correctly and the frequency is undisturbed. The hold timer is used to suppress frequency adjustments during the training and startup intervals described below. At the beginning of the interval the hold timer is set to the stepout threshold and decrements at one second intervals until reaching zero. However, the hold timer is forced to zero if the residual clock offset is less than 0.5 ms. When nonzero, the discipline algorithm uses a small time constant (equivalent to a poll exponent of 2), but does not adjust the frequency. Assuming that the frequency has been set to within 1 PPM, either from the frequency file or by the training interval described later, the clock is set to within 0.5 ms in less than 300 s.

9.13.4 Operating Intervals

The state machine operates in one of four nonoverlapping intervals.
Training interval  This interval is used at startup when the frequency file is not present at startup. It begins when the first update is received by the discipline algorithm and ends when an update is received following the stepout threshold. The clock phase is steered to the offset presented at the beginning of the interval, but without affecting the frequency. During the interval further updates are ignored. At the end of the interval the frequency is calculated as the phase change during the interval divided by the length of the interval. This generally results in a frequency error less than 0.5 PPM. Note that, if the intrinsic oscillator frequency error is large, the offset will in general have significant error. This is corrected during the subsequent startup interval.

Startup interval  This interval is used at startup to amortize the residual offset while not affecting the frequency. If the frequency file is present, it begins when the first update is received by the discipline. If not, it begins after the training interval. It ends when the hold timer decrements to zero or when the residual offset falls below 0.5 ms.

Step interval  This interval is used as a spike blanker during periods when the offsets exceed the step threshold. The interval continues as long as offsets are received that are greater than the step threshold, but ends when either an offset is received less than the step threshold or until the time since the last valid update exceeds the stepout threshold.

Sync Interval  This interval is implicit; that is, it is used when none of the above intervals are used.

9.13.5 State Transition Function

The state machine consists of five states. An event is created when an update is received by the discipline algorithm. Depending on the state and the offset magnitude, the machine performs some actions and transitions to the same or another state. Following is a short description of the states.

FSET - The frequency file is present  Load the frequency file, initialize the hold timer and continue in SYNC state.

NSET - The frequency file is not present  Initialize the hold timer and continue in FREQ state.

FREQ - Frequency training state  Disable the clock discipline until the time since the last update exceeds the stepout threshold. When this happens, calculate the frequency, initialize the hold counter and transition to SYNC state.

SPIK - Spike state  A update greater than the step threshold has occurred. Ignore the update and continue in this state as long as updates greater than the step threshold occur. If a valid update is received, continue in SYNC state. When the time since the last valid update was received exceeds the stepout threshold, step the system clock and continue in SYNC state.

SYNC - Ordinary clock discipline state  Discipline the system clock time and frequency using the hybrid phase/frequency feedback loop. However, do not discipline the frequency if the hold timer is nonzero.

9.14 Leap Second Processing

About every eighteen months the International Earth Rotation Service (IERS) issues a bulletin announcing the insertion of a leap second in the Universal Coordinated Time (UTC) timescale. Ordinarily, this happens at the end of the last day of June or December; but, in principle, it could happen at the end of any month. While these bulletins are available on the Internet at www.iers.org, advance notice of leap seconds is also available in signals broadcast from national time and frequency stations, in GPS signals and in telephone modem services. Many, but not all, reference clocks recognize these signals and many, but not all, drivers for them can decode the signals and set the leap bits in the timecode accordingly. This means that many, but not all, primary servers can pass on these bits in the NTP packet heard to dependent secondary servers and clients. Secondary servers can pass these bits to their dependents and so on throughout the NTP subnet.

A leap second is inserted following second 59 of the last minute of the day and becomes second 60 of that day. A leap second is deleted by omitting second 59 of the last minute of the day, although this has never happened and is highly
unlikely to happen in future. So far as is known, there are no provisions in the Unix or Windows libraries to account for this occasion other than to affect the conversion of an NTP datetamp or timestamp to conventional civil time.

When an update is received from a reference clock or downstratum server, the leap bits are inspected for all survivors of the cluster algorithm. If the number of survivors showing a leap bit is greater than half the total number of survivors, a pending leap condition exists until the end of the current month.

When no means are available to determine the leap bits from a reference clock or downstratum server, a leapseconds file can be downloaded from time.nist.gov and installed using the `leapfile` command. The file includes a list of historic leap seconds and the NTP time of insertion. It is parsed by the `ntpd` daemon at startup and the latest leap time saved for future reference. Each time the clock is set, the current time is compared with the last leap time. If the current time is later than the last leap time, nothing further is done. If earlier, the leap timer is initialized with the time in seconds until the leap time and counts down from there. When the leap timer is less than one month, a pending leap condition exists until the end of the current month. If the leapseconds file is present, the leap bits for reference clocks and downstratum servers are ignored.

If the precision time kernel support is available and enabled, at the beginning of the day of the leap event, the leap bits are set by the Unix `ntp_adjtime()` system call to arm the kernel for the leap at the end of the day. The kernel automatically inserts one second exactly at the time of the leap, after which the leap bits are turned off. If the kernel support is not availed or disabled, the leap is implemented as a crude hack by setting the clock back one second using the Unix `settimeofday()` system call, which effectively repeats the last second. Note however that in any case setting the time backwards by one second does not actually set the system clock backwards, but effectively stalls the clock for one second. These points are expanded in the white paper *The NTP Timescale and Leap Seconds*. If the leap timer is less than one day, the leap bits are set for dependent servers and clients.

As an additional feature when the NIST leap seconds file is installed, it is possible to determine the number of leap seconds inserted in UTC since UTC began on 1 January 1972. This represents the offset between International Atomic Time (TAI) and UTC. If the precision time kernel modifications are available and enabled, the TAI offset is available to application programs using the `antipasti()` system call. If the Autokey public-key cryptography feature is installed, the TAI offset is automatically propagated along with other cryptographic media to dependent servers and clients.
10.1 Undisciplined Local Clock

10.1.1 Synopsis

Address: 127.127.1.u
Reference ID: LOCL
Driver ID: LOCAL

10.1.2 Description

Note: We recommend against using this driver. A much more flexible replacement is described on the Orphan Mode page.

This driver was intended for use in an isolated network where no external source of synchronization such as a radio clock or modem is available. It allows a designated time server to act as a primary server to provide synchronization to other clients on the network. Pick a machine that has a good clock oscillator and configure it with this driver. Set the clock using the best means available, like eyeball-and-wristwatch. Then, point all the other machines at this one or use broadcast mode to distribute time.

Another application for this driver is if a particular server clock is to be used as the clock of last resort when all other normal synchronization sources have gone away. This is especially useful if that server has an ovenized oscillator. For this you would usually, but not necessarily, configure this driver at a stratum greater than any other likely sources of time, such as the default 5 for this driver, to prevent this driver taking over when legitimate sources elsewhere in the network are available. To further protect the Internet infrastructure from accidental or malicious exposure to this driver, the driver is disabled if another source is available and operating.

10.1.3 Monitor Data

No filegen clockstats monitor data are produced by this driver.

10.1.4 Fudge Factors

\texttt{time1 time} Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
\texttt{time2 time} Specifies the frequency offset calibration factor, in parts per million, with default 0.0.
\texttt{stratum number} Specifies the driver stratum, in decimal from 0 to 15, with default 5.
refid string  Specifies the driver reference identifier, an ASCII string from one to four characters, with default
LOCL.
flag1  0 | 1  Not used by this driver.
flag2  0 | 1  Not used by this driver.
flag3  0 | 1  Not used by this driver.
flag4  0 | 1  Not used by this driver.

10.1.5 Additional Information

Reference Clock Drivers

10.2 PSTI/Traconex 1020 WWV/WWVH Receiver

10.2.1 Synopsis

Address: 127.127.3.u
Reference ID: WWV
Driver ID: WWV_PST
Serial Port: /dev/wwvu; 9600 baud, 8-bits, no parity
Features: tty_clk

10.2.2 Description

This driver supports the PSTI 1010 and Traconex 1020 WWV/WWVH Receivers. No specific claim of accuracy is
made for these receiver, but actual experience suggests that 10 ms would be a conservative assumption.

The dipswitches should be set for 9600 bps line speed, 24-hour day-of-year format and UTC time zone. Automatic
correction for DST should be disabled. It is very important that the year be set correctly in the DIP-switches; otherwise,
the day of year will be incorrect after 28 April of a normal or leap year. As the there are only four dipswitches to set the
year and the base value of zero corresponds to 1986, years beyond 2001 recycle with the value of zero corresponding
to 2002. The propagation delay DIP-switches should be set according to the distance from the transmitter for both
WWV and WWVH, as described in the instructions. While the delay can be set only to within 11 ms, the fudge time1
parameter can be used for vernier corrections.

Using the poll sequence QTQDM, the response timecode is in three sections totalling 50 ASCII printing characters, as
concatenated by the driver, in the following format:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ah</td>
<td>AM/PM indicator (' ' for 24-hour mode)</td>
</tr>
<tr>
<td>h</td>
<td>hours</td>
</tr>
<tr>
<td>m</td>
<td>minutes</td>
</tr>
<tr>
<td>s</td>
<td>seconds</td>
</tr>
<tr>
<td>f</td>
<td>milliseconds</td>
</tr>
<tr>
<td>y</td>
<td>year (from DIPswitches)</td>
</tr>
<tr>
<td>d</td>
<td>day of month, month, day of year</td>
</tr>
<tr>
<td>s</td>
<td>daylight-saving indicator (' ' for 24-hour mode)</td>
</tr>
<tr>
<td>f</td>
<td>frequency enable (0 = all frequencies enabled)</td>
</tr>
<tr>
<td>r</td>
<td>baud rate (3 = 1200, 6 = 9600)</td>
</tr>
<tr>
<td>d</td>
<td>features indicator (@ = month/day display enabled)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on-time</th>
<th>first &lt;cr&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>hh:mm:ss.fff</td>
<td>yy/dd/mm/ddd&lt;cr&gt;</td>
</tr>
<tr>
<td>frdzycchSSFTttttuuuxx&lt;cr&gt;</td>
<td></td>
</tr>
</tbody>
</table>
The alarm condition is indicated by other than 8 at a, which occurs during initial synchronization and when received signal is lost for an extended period; unlock condition is indicated by other than 0000 in the tttt subfield.

### 10.2.3 Monitor Data

When enabled by the flag4 fudge flag, every received timecode is written as-is to the clockstats file.

### 10.2.4 Fudge Factors

- **time1 time**: Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
- **time2 time**: Not used by this driver.
- **stratum number**: Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- **refid string**: Specifies the driver reference identifier, an ASCII string from one to four characters, with default WWV.
- **flag1 0 | 1**: Not used by this driver.
- **flag2 0 | 1**: Not used by this driver.
- **flag3 0 | 1**: Not used by this driver.
- **flag4 0 | 1**: Not used by this driver.

### 10.2.5 Additional Information

Reference Clock Drivers

### 10.3 Spectracom WWVB/GPS Receivers

#### 10.3.1 Synopsis

- **Address**: 127.127.4.u
- **Reference ID**: WWVB
- **Driver ID**: WWVB_SPEC
- **Serial Port**: /dev/wwvbu; 9600 baud, 8-bits, no parity
- **Features**: Optional PPS signal processing, tty_clk
- **Requires**: Optional PPS signal processing requires the PPSAPI signal interface.
10.3.2 Description

This driver supports all known Spectracom radio and satellite clocks, including the Model 8170 and Netclock/2
WWVB Synchronized Clocks and the Netclock/GPS GPS Master Clock. The claimed accuracy of the WWVB clocks
is 100 ms relative to the broadcast signal. These clocks have proven a reliable source of time, except in some parts
of the country with high levels of conducted RF interference. With the GPS clock the claimed accuracy is 130 ns.
However, in most cases the actual accuracy is limited by the precision of the timecode and the latencies of the serial
interface and operating system.

The DIP switches on these clocks should be set to 24-hour display, AUTO DST off, data format 0 or 2 (see below)
and baud rate 9600. If this clock is used as the source for the IRIG Audio Decoder (refclock_irig.c in this
distribution), set the DIP switches for AM IRIG output and IRIG format 1 (IRIG B with signature control).

There are two timecode formats used by these clocks. Format 0, which is available with all clocks, and format 2,
which is available with all clocks except the original (unmodified) Model 8170.

Format 0 (22 ASCII printing characters):
<cr><lf>i ddd hh:mm:ss TZ=zz<cr><lf>
on-time = first <cr>
i = synchronization flag (' ' = in synch, '?' = out synch)
hh:mm:ss = hours, minutes, seconds

The alarm condition is indicated by other than ' ' at i, which occurs during initial synchronization and when received
signal is lost for about ten hours.

Format 2 (24 ASCII printing characters):
l<cr><lf>i qyy ddd hh:mm:ss.fff ld

on-time = <cr>
i = synchronization flag (' ' = in synch, '?' = out synch)
q = quality indicator (' ' = locked, 'A'...'D' = unlocked)
yy = year (as broadcast)
ddd = day of year
hh:mm:ss.fff = hours, minutes, seconds, milliseconds

The alarm condition is indicated by other than ' ' at i, which occurs during initial synchronization and when received
signal is lost for about ten hours. The unlock condition is indicated by other than ' ' at q.

The q is normally ' ' when the time error is less than 1 ms and a character in the set A...D when the time error is
less than 10, 100, 500 and greater than 500 ms respectively. The l is normally ' ', but is set to L early in the month of
an upcoming UTC leap second and reset to ' ' on the first day of the following month. The d is set to S for standard
time S, I on the day preceding a switch to daylight time, D for daylight time and O on the day preceding a switch to
standard time. The start bit of the first <cr> is synchronized to the indicated time as returned.

This driver does not need to be told which format is in use - it figures out which one from the length of the message.
A three-stage median filter is used to reduce jitter and provide a dispersion measure. The driver makes no attempt to
correct for the intrinsic jitter of the radio itself, which is a known problem with the older radios.
10.3.3 PPS Signal Processing

When PPS signal processing is enabled, and when the system clock has been set by this or another driver and the PPS signal offset is within 0.4 s of the system clock offset, the PPS signal replaces the timecode for as long as the PPS signal is active. If for some reason the PPS signal fails for one or more poll intervals, the driver reverts to the timecode. If the timecode fails for one or more poll intervals, the PPS signal is disconnected.

10.3.4 Monitor Data

The driver writes each timecode as received to the `clockstats` file. When enabled by the `flag4` fudge flag, a table of quality data maintained internally by the Netclock/2 is retrieved and written to the `clockstats` file when the first timecode message of a new day is received.

10.3.5 Fudge Factors

- `time1 time` Specifies the PPS time offset calibration factor, in seconds and fraction, with default 0.0.
- `time2 time` Specifies the serial time offset calibration factor, in seconds and fraction, with default 0.0.
- `stratum number` Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- `refid string` Specifies the driver reference identifier, an ASCII string from one to four characters, with default `WWVB`.
- `flag1 0 | 1` Disable PPS signal processing if 0 (default); enable PPS signal processing if 1.
- `flag2 0 | 1` If PPS signal processing is enabled, capture the pulse on the rising edge if 0 (default); capture on the falling edge if 1.
- `flag3 0 | 1` If PPS signal processing is enabled, use the `ntpd` clock discipline if 0 (default); use the kernel discipline if 1.
- `flag4 0 | 1` Enable verbose `clockstats` recording if set.

10.4 TrueTime GPS/GOES/OMEGA/WWV Receivers

10.4.1 Synopsis

Address: 127.127.5.u
Reference ID: GPS, OMEGA, GOES, WWV
Driver ID: TRUETIME
Serial Port: `/dev/trueu`; 9600 baud, 8-bits, no parity
Features: `tty_clk`

10.4.2 Description

This driver supports several models of Kinematics/TrueTime timing receivers, including 468-DC MK III GOES Synchronized Clock, GPS- DC MK III and GPS/TM-TMD GPS Synchronized Clock, XL-DC (a 151-602-210, reported by the driver as a GPS/TM-TMD), GPS-800 TCU (an 805-957 with the RS232 Talker/Listener module), OM-DC OMEGA Synchronized Clock, the TL-3 WWV receiver, and very likely others in the same model families that use the same timecode formats.
Most of this code is originally from refclock_wwvb.c with thanks. It has been so mangled that wwvb is not a recognizable ancestor.

**Timecode format:** \texttt{ADDD:HH:MM:SSQCL}  
A - control A (this is stripped before we see it)  
Q - Quality indication  
(see below)  
C - Carriage return  
L - Line feed

Quality codes indicate possible error of:

- **468-DC GOES Receiver:** ? +/- 500 milliseconds # +/- 50 milliseconds  
  * +/- 5 milliseconds . +/- 1 millisecond space less than 1 millisecond

- **GPS-TM/TMD Receiver:** ? +/- 500 milliseconds # +/- 50 milliseconds  
  * +/- 5 milliseconds . +/- 1 millisecond space less than 1 millisecond

- **OM-DC OMEGA Receiver:** > +/- 5 seconds  
  ? +/- 500 milliseconds # +/- 50 milliseconds  
  * +/- 5 milliseconds . +/- 1 millisecond

- **TL-3 WWV Receiver:** ? receiver is unlocked space +/- 5 milliseconds

### 10.4.3 Notes on 468-DC and OMEGA receiver:

Send the clock a \texttt{R} or \texttt{C} and once per second a timestamp will appear. Send a \texttt{R} to get the satellite position once (GOES only).

### 10.4.4 Notes on the 468-DC receiver:

Since the old east/west satellite locations are only historical, you can’t set your clock propagation delay settings correctly and still use automatic mode. The manual says to use a compromise when setting the switches. This results in significant errors. The solution; use fudge time1 and time2 to incorporate corrections. If your clock is set for 50 and it should be 58 for using the west and 46 for using the east, use the line

\begin{verbatim}
fudge 127.127.5.0 time1 +0.008 time2 -0.004
\end{verbatim}

This corrects the 4 milliseconds advance and 8 milliseconds retard needed. The software will ask the clock which satellite it sees.

The PCL720 from PC Labs has an Intel 8253 look-alike, as well as a bunch of TTL input and output pins, all brought out to the back panel. If you wire a PPS signal (such as the TTL PPS coming out of a GOES or other Kinematics/Truetime clock) to the 8253’s GATE0, and then also wire the 8253’s OUT0 to the PCL720’s INPUT3.BIT0, then we can read CTR0 to get the number of microseconds since the last PPS upward edge, mediated by reading OUT0 to find out if the counter has wrapped around (this happens if more than 65535us (65ms) elapses between the PPS event and our being called.)

### 10.4.5 Notes on the TL-3 receiver:

The mini-DIN RS-232 port uses the Apple pinout.

Send the clock ST1 to turn on continuous (1/sec) timecodes. You can also enable “mode C” via the front panel. ST0 turns off this mode.

QV will return the firmware revision (and is useful in identifying this clock.)
QW will return its weekly signal log, useful if you’re testing antennas. You may wish to turn the loss interval down from 4h (04) to 1h (01), so the receiver declares itself unlocked sooner. When in holdover, drift can be on the order of 10 ms/hr since there is no high quality reference oscillator.

10.4.6 Monitor Data

When enabled by the flag4 fudge flag, every received timecode is written as-is to the clockstats file.

10.4.7 Fudge Factors

\texttt{time1 time} Specifies the time offset calibration factor, in seconds and fraction, to be used for the West satellite, with default 0.0.

\texttt{time2 time} Specifies the time offset calibration factor, in seconds and fraction, to be used for the East satellite, with default 0.0.

\texttt{stratum number} Specifies the driver stratum, in decimal from 0 to 15, with default 0.

\texttt{refid string} Specifies the driver reference identifier, an ASCII string from one to four characters, with default TRUE.

\texttt{flag1 0 | 1} Silence the clock side of ntpd, just reading the clock without trying to write to it.

\texttt{flag2 0 | 1} Generate a debug file /tmp/true%d.

\texttt{flag3 0 | 1} Not used by this driver.

\texttt{flag4 0 | 1} Enable verbose clockstats recording if set.

10.4.8 Additional Information

Reference Clock Drivers

10.5 IRIG Audio Decoder

10.5.1 Synopsis

Address: 127.127.6.u
Reference ID: IRIG
Driver ID: IRIG_AUDIO
Audio Device: /dev/audio and /dev/audioctl

10.5.2 Description

This driver synchronizes the computer time using the Inter-Range Instrumentation Group (IRIG) standard time distribution signal. This signal is generated by several radio clocks, including those made by Arbiter, Austron, Bancomm, Odetics, Spectracom, Symmetricom and TrueTime, among others, although it is often an add-on option. The signal is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC.

The driver requires an audio codec or sound card with sampling rate 8 kHz and \(\mu\)-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating
systems, including Solaris, FreeBSD and Linux, among others. In this implementation, only one audio driver and codec can be supported on a single machine. In order to assure reliable signal capture, the codec frequency error must be less than 250 PPM (.025 percent). If necessary, the tinker codec configuration command can be used to bracket the codec frequency to this range.

For proper operation the IRIG signal source should be configured for analog signal levels, not digital TTL levels. In most radios the IRIG signal is driven ±10 V behind 50 Ohms. In such cases the cable should be terminated at the line-in port with a 50-Ohm resistor to avoid overloading the codec. Where feasible, the IRIG signal source should be operated with signature control so that, if the signal is lost or mutilated, the source produces an unmodulated signal, rather than possibly random digits. The driver automatically rejects the data and declares itself unsynchronized in this case. Some devices, in particular Spectracom radio/satellite clocks, provide additional year and status indication; other devices may not.

In general and without calibration, the driver is accurate within 500 μs relative to the IRIG time. After calibrating relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is less than 20 μs with standard deviation 10 μs. Most of this is due to residuals after filtering and averaging the raw codec samples, which have an inherent jitter of 125 μs. The processor load due to the driver is 0.6 percent on the P4.

However, be acutely aware that the accuracy with Solaris 2.8 and beyond has been seriously degraded to the order of several milliseconds. The Sun kernel driver has a sawtooth modulation with amplitude over 5 ms P-P and period 5.5 s. This distortion is especially prevalent with Sun Blade 1000 and possibly other systems.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.

This driver incorporates several features in common with other audio drivers such as described in the Radio CHU Audio Demodulator/Decoder and the Radio WWV/H Audio Demodulator/Decoder pages. They include automatic gain control (AGC), selectable audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hookup, debugging and monitoring, see the Reference Clock Audio Drivers page.

### 10.5.3 Technical Overview

The IRIG signal format uses an amplitude-modulated carrier with pulse-width modulated data bits. For IRIG-B, the carrier frequency is 1000 Hz and bit rate 100 b/s; for IRIG-E, the carrier frequency is 100 Hz and bit rate 10 b/s. While IRIG-B provides the best accuracy, generally within a few tens of microseconds relative to IRIG time, it can also generate a significant processor load with older workstations. Generally, the accuracy with IRIG-E is about ten times worse than IRIG-B, but the processor load is somewhat less. Technical details about the IRIG formats can be found in IRIG Standard 200-98.

The driver processes 8000-Hz μ-law companded samples using separate signal filters for IRIG-B and IRIG-E, a comb filter, envelope detector and automatic threshold corrector. An infinite impulse response (IIR) 1000-Hz bandpass filter is used for IRIG-B and an IIR 130-Hz lowpass filter for IRIG-E. These are intended for use with noisy signals, such as might be received over a telephone line or radio circuit, or when interfering signals may be present in the audio passband. The driver determines which IRIG format is in use by sampling the amplitude of each filter output and selecting the one with maximum signal.

Cycle crossings relative to the corrected slice level determine the width of each pulse and its value - zero, one or position identifier (PI). The data encode ten characters (20 BCD digits) which determine the second, minute, hour and day of the year and with some IRIG generators the year and synchronization condition. The comb filter exponentially averages the corresponding samples of successive baud intervals in order to reliably identify the reference carrier cycle.

A type-II phase-lock loop (PLL) performs additional integration and interpolation to accurately determine the zero crossing of that cycle, which determines the reference timestamp. A pulse-width discriminator demodulates the data pulses, which are then encoded as the BCD digits of the timecode. The timecode and reference timestamp are updated once each second with IRIG-B (ten seconds with IRIG-E) and local clock offset samples saved for later processing. At poll intervals of 64 s, the saved samples are processed by a median filter and used to update the system clock.
### 10.5.4 Monitor Data

The timecode format used for debugging and data recording includes data helpful in diagnosing problems with the IRIG signal and codec connections. The driver produces one line for each timecode in the following format:

```
    00 00 98 23 19:26:52 2782 143 0.694 10 0.3 66.5 3094572411.00027
```

If clockstats is enabled, the most recent line is written to the clockstats file every 64 s. If verbose recording is enabled (fudge flag 4) each line is written as generated.

The first field contains the error flags in hex, where the hex bits are interpreted as below. This is followed by the year of century, day of year and time of day. Note that the time of day is for the previous minute, not the current time. The status indicator and year are not produced by some IRIG devices and appear as zeros. Following these fields are the carrier amplitude (0-3000), codec gain (0-255), modulation index (0-1), time constant (4-10), carrier phase error (0±0.5) and carrier frequency error (PPM). The last field is the on-time timestamp in NTP format.

The error flags are defined as follows in hex:

- **x01** Low signal. The carrier amplitude is less than 100 units. This is usually the result of no signal or wrong input port.
- **x02** Frequency error. The codec frequency error is greater than 250 PPM. This may be due to wrong signal format or (rarely) defective codec.
- **x04** Modulation error. The IRIG modulation index is less than 0.5. This is usually the result of an overdriven codec, wrong signal format or wrong input port.
- **x08** Frame synch error. The decoder frame does not match the IRIG frame. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal. It may also be the result of an IRIG signature check which indicates a failure of the IRIG signal synchronization source.
- **x10** Data bit error. The data bit length is out of tolerance. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal.
- **x20** Seconds numbering discrepancy. The decoder second does not match the IRIG second. This is usually the result of an overdriven codec, wrong signal format or noisy IRIG signal.
- **x40** Codec error (overrun). The machine is not fast enough to keep up with the codec.
- **x80** Device status error (Spectracom).

### 10.5.5 Fudge Factors

- **time1** time  Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
- **time2** time  Not used by this driver.
- **stratum** number  Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- **refid** string  Specifies the driver reference identifier, an ASCII string from one to four characters, with default [IRIG](#).
- **flag1** 0 | 1  Not used by this driver.
- **flag2** 0 | 1  Specifies the microphone port if set to zero or the line-in port if set to one. It does not seem useful to specify the compact disc player port.
- **flag3** 0 | 1  Enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.
- **flag4** 0 | 1  Enable verbose clockstats recording if set.
10.6 Radio CHU Audio Demodulator/Decoder

10.6.1 Synopsis

Address: 127.127.7.\textit{u}
Reference ID: CHU
Driver ID: CHU
Modem Port: /dev/chuu; 300 baud, 8-bits, no parity
Autotune Port: /dev/icom; 1200/9600 baud, 8-bits, no parity
Audio Device: /dev/audio and /dev/audioctl

10.6.2 Description

This driver synchronizes the computer time using shortwave radio transmissions from Canadian time/frequency station CHU in Ottawa, Ontario. CHU transmissions are made continuously on 3.330, 7.850 and 14.670 MHz in upper sideband, compatible AM mode. An ordinary shortwave receiver can be tuned manually to one of these frequencies or, in the case of ICOM receivers, the receiver can be tuned automatically as propagation conditions change throughout the day and season.

The driver can be compiled to use either an audio codec or soundcard, or a Bell 103-compatible, 300-b/s modem or modem chip, as described on the Pulse-per-second (PPS) Signal Interfacing page. If compiled for a modem, the driver uses it to receive the radio signal and demodulate the data. If compiled for the audio codec, it requires a sampling rate of 8 kHz and \( \mu \)-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating systems, including Solaris, FreeBSD and Linux, among others. The radio is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC. In this implementation, only one audio driver and codec can be supported on a single machine.

In general and without calibration, the driver is accurate within 1 ms relative to the broadcast time when tracking a station. However, variations up to 0.3 ms can be expected due to diurnal variations in ionospheric layer height and ray geometry. In Newark DE, 625 km from the transmitter, the predicted one-hop propagation delay varies from 2.8 ms in sunlight to 2.6 ms in moonlight. When not tracking the station the accuracy depends on the computer clock oscillator stability, ordinarily better than 0.5 PPM.

After calibration relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is generally within 0.2 ms short-term with 0.4 ms jitter. The long-term mean offset varies up to 0.3 ms due to propagation path geometry variations. The processor load due to the driver is 0.4 percent on the P4.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.

This driver incorporates several features in common with other audio drivers such as described in the Radio WWV/H Audio Demodulator/Decoder and the IRIG Audio Decoder pages. They include automatic gain control (AGC), selectable audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hookup, debugging and monitoring, see the Reference Clock Audio Drivers page.

10.6.3 Technical Overview

The driver processes 8-kHz \( \mu \)-law companded codec samples using maximum-likelihood techniques which exploit the considerable degree of redundancy available in each broadcast message or burst. As described below, every character is sent twice and, in the case of format A bursts, the burst is sent eight times every minute. The single format B burst is considered correct only if every character matches its repetition in the burst. For the eight format A bursts, a majority
decoder requires more than half of the 16 repetitions for each digit decode to the same value. Every character in every burst provides an independent timestamp upon arrival with a potential total of 60 timestamps for each minute.

The CHU timecode format is described on the CHU website. A timecode is assembled when all bursts have been received in each minute. The timecode is considered valid and the clock set when at least one valid format B burst has been decoded and the majority decoder declares success. Once the driver has synchronized for the first time, it will appear reachable and selectable to discipline the system clock. It is normal on occasion to miss a minute or two due to signal fades or noise. If eight successive minutes are missed, the driver is considered unreachable and the system clock will free-wheel at the latest determined frequency offset. Since the signals are almost always available during some period of the day and the NTP clock discipline algorithms are designed to work well even with long intervals between updates, it is unlikely that the system clock will drift more than a few milliseconds during periods of signal loss.

10.6.4 Baseband Signal Processing

The program consists of four major parts: the DSP modem, maximum-likelihood UART, burst assembler and majority decoder. The DSP modem demodulates Bell 103 modem answer-frequency signals; that is, frequency-shift keyed (FSK) tones of 2225 Hz (mark) and 2025 Hz (space). It consists of a 500-Hz bandpass filter centered on 2125 Hz followed by a limiter/discriminator and raised-cosine lowpass filter optimized for the 300-b/s data rate.

The maximum likelihood UART is implemented using a set of eight 11-stage shift registers, one for each of eight phases of the 300-b/s bit clock. At each phase a new baseband signal from the DSP modem is shifted into the corresponding register and the maximum and minimum over all 11 samples computed. This establishes a span (difference) and slice level (average) over all 11 stages. For each stage, a signal level above the slice is a mark (1) and below that is a space (0). A quality metric is calculated for each register with respect to the slice level and the a-priori signal consisting of a start bit (space), eight arbitrary information bits and two stop bits (mark).

The shift registers are processed in round-robin order as the phases of each bit arrive. At the end of each bit all eight phases are searched for valid framing bits, sufficient span and best metric. The best candidate found in this way represents the maximum-likelihood character. The process then continues for all ten characters in the burst.

The burst assembler processes characters either from the maximum-likelihood UART or directly from the serial port as configured. A burst begins when a character is received and is processed after a timeout interval when no characters are received. If the interval between characters is greater than two characters, but less than the timeout interval, the burst is rejected as a runt and a new burst begun. As each character is received, a timestamp is captured and saved for later processing.

A valid burst consists of ten characters in two replicated five-character blocks, each block representing ten 4-bit BCD digits. The format B blocks sent in second 31 contain the year and other information in ten digits. The eight format A blocks sent in seconds 32-39 contain the timecode in ten digits, the first of which is a framing code (6). The burst assembler must deal with cases where the first character of a format A burst is lost or is noise. This is done using the framing codes to correct the discrepancy, either one character early or one character late.

The burst distance is incremented by one for each bit in the first block that matches the corresponding bit in the second block and decremented by one otherwise. In a format B burst the second block is bit-inverted relative to the first, so a perfect burst of five 8-bit characters has distance -40. In a format A burst the two blocks are identical, so a perfect burst has distance +40. Format B bursts must be perfect to be acceptable; however, format A bursts, which are further processed by the majority decoder, are acceptable if the distance is at least 28.

10.6.5 Majority Decoder

Each minute of transmission includes eight format A bursts containing two timecodes for each second from 32 through 39. The majority decoder uses a decoding matrix of ten rows, one for each digit position in the timecode, and 16 columns, one for each 4-bit code combination that might be decoded at that position. In order to use the character
timestamps, it is necessary to reliably determine the second number of each burst. In a valid burst, the last digit of the
two timecodes in the burst must match and the value must be in the range 2-9 and greater than in the previous burst.

As each digit of a valid burst is processed, the value at the row corresponding to the digit position in the timecode and
column corresponding to the code found at that position is incremented. At the end of the minute, each row of the
decoding matrix encodes the number of occurrences of each code found at the corresponding position.

The maximum over all occurrences at each digit position is the distance for that position and the corresponding code
is the maximum-likelihood digit. If the distance is not more than half the total number of occurrences, the decoder
assumes a soft error and discards all information collected during the minute. The decoding distance is defined as the
sum of the distances over the first nine digits; the tenth digit varies over the seconds and is uncounted.

The result of the majority decoder is a nine-digit timecode representing the maximum-likelihood candidate for the
transmitted timecode in that minute. Note that the second and fraction within the minute are always zero and that
the actual reference point to calculate timestamp offsets is backdated to the first second of the minute. At this point
the timecode block is reformatted and the year, days, hours and minutes extracted along with other information from
the format B burst, including DST state, DUT1 correction and leap warning. The reformattting operation checks
the timecode for invalid code combinations that might have been left by the majority decoder and rejects the entire
timecode if found.

If the timecode is valid, it is passed to the reference clock interface along with the backdated timestamps accumulated
over the minute. A perfect set of eight bursts could generate as many as 80 timestamps, but the maximum the interface
can handle is 60. These are processed using a median filter and trimmed-mean average, so the resulting system clock
rection is usually much better than would otherwise be the case with radio noise, UART jitter and occasional burst
erons.

**10.6.6 Autotune**

The driver includes provisions to automatically tune the radio in response to changing radio propagation conditions
throughout the day and night. The radio interface is compatible with the ICOM CI-V standard, which is a bidirectional
serial bus operating at TTL levels. The bus can be connected to a standard serial port using a level converter such as
the CT-17. Further details are on the Reference Clock Audio Drivers page.

If specified, the driver will attempt to open the device /dev/icom and, if successful will tune the radio to 3.331
MHz. The 1-kHz offset is useful with a narrowband SSB filter where the passband includes the carrier and modem
signals. However, the driver is liberal in what it assumes of the configuration. If the /dev/icom link is not present
or the open fails or the CI-V bus is inoperative, the driver continues in single-frequency mode.

As long as no bursts are received, the driver cycles over the three frequencies in turn, one minute for each station.
When bursts are received from one or more stations, the driver operates in a five-minute cycle. During the first four
minutes it tunes to the station with the highest metric. During the last minute it alternates between the other two
stations in turn in order to measure the metric.

**10.6.7 Debugging Aids**

The most convenient way to track the program status is using the ntpq program and the clockvar command. This
displays the last determined timecode and related status and error counters, even when the program is not discipline
the system clock. If the debugging trace feature (-d on the ntpd command line) is enabled, the program produces
detailed status messages as it operates. If the fudge flag 4 is set, these messages are written to the clockstats
file. All messages produced by this driver have the prefix chu for convenient filtering with the Unix grep command.

With debugging enabled the driver produces messages in the following formats: A single message beginning with
chuB is produced for each format B burst received in second 31, while eight messages beginning with chuA are
produced for each format A burst received in seconds 32 through 39 of the minute. The first four fields are
stat sig n b
where \texttt{stat} is the status code, \texttt{sig} the character span, \texttt{n} the number of characters in the burst (9-11) and \texttt{b} the burst distance (0-40). Good bursts will have spans of a 800 or more and the other numbers near the top of the range specified. See the source for the interpretation of the remaining data in the burst. Note that each character of the burst is encoded as two digits in nibble-swapped order.

If the CI-V interface for ICOM radios is active, a debug level greater than 1 will produce a trace of the CI-V command and response messages. Interpretation of these messages requires knowledge of the CI-V protocol, which is beyond the scope of this document.

### 10.6.8 Monitor Data

When enabled by the \texttt{filegen} facility, every received timecode is written to the \texttt{clockstats} file in the following format:

```
sq yyyy ddd hh:mm:ss lw dst du lset agc rfrq bcnt dist tsmp
```

- \texttt{s} sync indicator
- \texttt{q} quality character
- \texttt{yyyy} Gregorian year
- \texttt{ddd} day of year
- \texttt{hh} hour of day
- \texttt{mm} minute of hour
- \texttt{ss} second of minute
- \texttt{lw} leap second warning
- \texttt{dst} DST state
- \texttt{dut} DUT sign and magnitude in deciseconds
- \texttt{lset} minutes since last set
- \texttt{agc} audio gain (0-255)
- \texttt{ident} CHU identifier code
- \texttt{dist} decoder distance
- \texttt{tsmp} timestamps captured

The fields beginning with \texttt{year} and extending through \texttt{dut} are decoded from the received data and are in fixed-length format. The \texttt{agc} and \texttt{lset} fields, as well as the following driver-dependent fields, are in variable-length format.

- \texttt{s} The sync indicator is initially \texttt{?} before the clock is set, but turns to space when the clock has been correctly set.
- \texttt{q} The quality character is a four-bit hexadecimal code showing which alarms have been raised during the most recent minute. Each bit is associated with a specific alarm condition according to the following:
  - 8 Timestamp alarm. Fewer than 20 timestamps have been determined.
  - 4 Decoder alarm. A majority of repetitions for at least one digit of the timecode fails to agree.
  - 2 Format alarm. One or more bursts contained invalid data or was improperly formatted.
  - 1 Frame alarm. One or more bursts was improperly framed or contained too many repetition errors.

The timestamp and decoder alarms are fatal; the data accumulated during the minute are not used to set the clock. The format and fram alarm are nonfatal; only the data in the burst are discarded.

- \texttt{yyyy ddd hh:mm:ss} The timecode format itself is self explanatory. Note that the Gregorian year is decoded directly from the transmitted timecode.
- \texttt{lw} The leap second warning is normally space, but changes to \texttt{L} if a leap second is to occur at the end of the month.
- \texttt{dst} The DST code for Canada encodes the state for all provinces. It is encoded as two hex characters.
- \texttt{dut} The DUT sign and magnitude shows the current UT1 offset relative to the displayed UTC time, in deciseconds. It is encoded as one digit preceeded by sign.
**lset** Before the clock is set, this is the number of minutes since the program was started; after the clock is set, this is the number of minutes since the time was last verified relative to the broadcast signal.

**agc** The audio gain shows the current codec gain setting in the range 0 to 255. Ordinarily, the receiver audio gain control should be set for a value midway in this range.

**ident** The CHU identifier CHU followed by the current radio frequency code, if the CI-V interface is active, or CHU if not. The radio frequency is encoded as 0 for 3.330 MHz, 1 for 7.850 MHz and 2 for 14.670 MHz.

**dist** The decoding distance determined during the most recent minute bursts were received. The values range from 0 to 160, with the higher values indicating better signals. The decoding algorithms require the distance at least 50; otherwise all data in the minute are discarded.

**tsmp** The number of timestamps determined during the most recent minute bursts were received. The values range from 0 to 60, with the higher values indicating better signals. The decoding algorithms require at least 20 timestamps in the minute; otherwise all data in the minute are discarded.

### 10.6.9 Fudge Factors

**time1 time** Specifies the propagation delay for CHU (45:18N 75:45N), in seconds and fraction, with default 0.0.

**time2 time** Not used by this driver.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default CHU.

**flag1 0 | 1** Not used by this driver.

**flag2 0 | 1** When the audio driver is compiled, this flag selects the audio input port, where 0 is the mike port (default) and 1 is the line-in port. It does not seem useful to select the compact disc player port.

**flag3 0 | 1** When the audio driver is compiled, this flag enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.

**flag4 0 | 1** Enable verbose clockstats recording if set.

### 10.7 Generic Reference Driver

#### 10.7.1 Synopsis

Address: 127.127.8.u
Reference ID: PARSE
Driver ID: GENERIC
Serial Port: /dev/refclock-u; TTY mode according to clock type
PPS device: /dev/refclockpps-u; alternate PPS device (if not available via the serial port)

#### 10.7.2 Description

The PARSE driver supports 20 different clock types/configurations. PARSE is actually a multi-clock driver.
The actual receiver status is mapped into various synchronization states generally used by receivers. The driver is configured to interpret the time codes of Meinberg DCF77 AM receivers, DCF77 FM receivers, Meinberg GPS16x/17x receivers, Trimble SV6 GPS, ELV DCF7000, Schmid, Wharton 400A and low cost receivers (see list below).

The reference clock support in NTP contains the necessary configuration tables for those receivers. In addition to supporting several different clock types and up to 4 devices, the processing 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 ntpd 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.

### 10.7.3 Timecode variables listed by ntpq (8)

The ntpq program can read and display several clock variables. These hold the following information:

**refclock_format** A qualification of the decoded time code format.

**refclock_states** The overall running time and the accumulated times for the clock event states.

**refclock_status** Lists the currently active receiver flags. Additional feature flags for the receiver are optionally listed in parentheses.

**refclock_time** The local time with the offset to UTC (format HHMM).

**timecode** The actual time code.

If PPS information is present, additional variables are available:

**refclock_ppsskew** The difference between the RS-232-derived timestamp and the PPS timestamp.

**refclock_ppstime** The PPS timestamp.

### 10.7.4 Supported Devices

Currently, twenty-four clock types are supported by the PARSE driver and up to four (devices /dev/refclock-0 - /dev/refclock-3) of these clocks may be operational at any one time.

A note on the implementations:

- These implementations were mainly done without actual access to the hardware, thus not all implementations provide full support. The development was done with the help of many kind souls who had the hardware and kindly lent me their time and patience during the development and debugging cycle. Thus for continued support and quality, direct access to the receivers is a big help. Nevertheless I am not prepared to buy these reference clocks - donations to (kardel <AT> ntp.org) are welcome as long as they work within Europe 8-).

Verified implementations are:

- RAWDCF variants
  These variants have been tested for correct decoding with my own homegrown receivers. Interfacing with specific commercial products may involve some fiddling with cables. In particular, commercial RAWDCF receivers have a seemingly unlimited number of ways to draw power from the RS-232 port and to encode the DCF77 datastream. You are mainly on your own here unless I have a sample of the receiver.

- Meinberg clocks
These implementations have been verified by the Meinberg people themselves and I have access to one of these clocks.

– Schweitzer Engineering Laboratories SEL-240x clocks

This implementation was provided and verified by SEL and Network Time Foundation has an SEL-2407 in one of its development labs.

The pictures below have been taken from and are linked to the vendors’ web pages.

• **server 127.127.8.0-3 mode 0**
  Meinberg PZF5xx receiver family (FM demodulation/TCXO / 50µs)

• **server 127.127.8.0-3 mode 1**
  Meinberg PZF5xx receiver family (FM demodulation/OCXO / 50µs)

• **server 127.127.8.0-3 mode 2**
  Meinberg DCF C51 receiver and similar (AM demodulation / 4ms)
This mode expects the Meinberg standard time string format with 9600/7E2.

**Note:** mode 2 must also be used for Meinberg PCI cards under Linux, e.g. the GPS PCI card or the DCF77 PCI card. Please note the Meinberg Linux driver must be installed. That driver emulates a refclock device in order to allow ntpd to access those cards. For details, please refer to the README file that comes with the Meinberg driver package.

- **server 127.127.8.0-3 mode 3**
  - ELV DCF7000 (sloppy AM demodulation / 50ms)
- **server 127.127.8.0-3 mode 4**
  - Walter Schmid DCF receiver Kit (AM demodulation / 1ms)
- **server 127.127.8.0-3 mode 5**
  - RAW DCF77 100/200ms pulses (Conrad DCF77 receiver module / 5ms)
- **server 127.127.8.0-3 mode 6**
  - RAW DCF77 100/200ms pulses (TimeBrick DCF77 receiver module / 5ms)
- **server 127.127.8.0-3 mode 7**
  - Meinberg GPS16x/GPS17x receivers (GPS / <<1μs)
This mode expects either the University of Erlangen time string format or the Meinberg standard time string format at 19200/8N1.

The University of Erlangen format is preferred. Newer Meinberg GPS receivers can be configured to transmit that format; for older devices, a special firmware version may be available.

In this mode some additional GPS receiver status information is also read. However, this requires a point-to-point connection. Mode 18 should be used if the device is accessed by a multidrop connection.

Note: mode 7 must not be used with Meinberg PCI cards; use mode 2 instead.

- server 127.127.0.3 mode 8
  IGEL clock

- server 127.127.0.3 mode 9
Trimble SVeeSix GPS receiver TAIP protocol (GPS / <<1µs)

- server 127.127.8.0-3 mode 10

Trimble SVeeSix GPS receiver TSIP protocol (GPS / <<1µs) (no kernel support yet)

- server 127.127.8.0-3 mode 11

Radiocode Clocks Ltd RCC 8000 Intelligent Off-Air Master Clock support

- server 127.127.8.0-3 mode 12

HOPF Funkuhr 6021

- server 127.127.8.0-3 mode 13

Diem’s Computime Radio Clock

- server 127.127.8.0-3 mode 14

RAWDCF receiver (DTR=high/RTS=low)

- server 127.127.8.0-3 mode 15

WHARTON 400A Series Clocks with a 404.2 Serial Interface

- server 127.127.8.0-3 mode 16

RAWDCF receiver (DTR=low/RTS=high)

- server 127.127.8.0-3 mode 17

VARITEXT Receiver (MSF)
server 127.127.8.0-3 mode 18

Meinberg GPS16x/GPS17x receivers (GPS / $<1\mu$s)

This mode works without additional data communication (version, GPS status etc.) and thus should be used with multidrop, heterogeneous multiclient operation.

Note: mode 18 must not be used with Meinberg PCI cards, use mode 2 instead.

• server 127.127.8.0-3 mode 19

Gude Analog- und Digitalsystem GmbH ‘Expert mouseCLOCK USB v2.0’

• server 127.127.8.0-3 mode 20

RAWDCF receiver similar to mode 14, but operating @ 75 baud (DTR=high/RTS=low)

Driving the DCF clocks at 75 baud may help to get them to work with a bunch of common USB serial converters, that do 75 but cannot do 50 baud at all, e.g. those based on Prolific PL2303.

• server 127.127.8.0-3 mode 21

RAWDCF receiver similar to mode 16, but operating @ 75 baud (DTR=low/RTS=high)

See comment from mode 20 clock.

• server 127.127.8.0-3 mode 22

MEINBERG, mode 2 but with POWERUP trust

• server 127.127.8.0-3 mode 23

MEINBERG, mode 7 but with POWERUP trust

• server 127.127.8.0-3 mode 24

Schweitzer Engineering Laboratories

Actual data formats and setup requirements of the various clocks can be found in NTP PARSE clock data formats.

10.7.5 Operation

The reference clock support software 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 is a function of clock type.

Raw DCF77 pulses can be fed via a level converter to the RXD pin of an RS-232 serial port (pin 3 of a 25-pin connector or pin 2 of a 9-pin connector). The telegrams are decoded and used for synchronization. DCF77 AM receivers can be bought for as little as $25. The accuracy is dependent on the receiver and is somewhere between 2ms (expensive) and 10ms (cheap). Synchronization ceases when reception of the DCF77 signal deteriorates, since no backup oscillator is available as usually found in other reference clock receivers. So it is important to have a good place for the DCF77 antenna. During transmitter shutdowns you are out of luck unless you have other NTP servers with alternate time sources available.

In addition to the PPS loopfilter control, a true PPS hardware signal can be utilized via the PPSAPI interface. PPS pulses are usually fed via a level converter to the DCD pin of an RS-232 serial port (pin 8 of a 25-pin connector or pin 1 of a 9-pin connector). To select PPS support, the mode parameter is the mode value as above plus 128. If 128 is not added to the mode value, PPS will be detected to be available but will not be used.
10.7.6 Hardware PPS support

For PPS to be used, add 128 to the mode parameter.

If the PPS signal is fed in from a device different from the device providing the serial communication (/dev/refclock-{0..3}), this device is configured as /dev/refclockpps-{0..3}. This allows the PPS information to be fed in e.g. via the parallel port (if supported by the underlying operation system) and the date/time telegrams to be handled via the serial port.

10.7.7 Monitor Data

Clock state statistics are written hourly to the syslog service. Online information can be found by examining the clock variables via the ntpq cv command.

Some devices have quite extensive additional information (GPS16x/GPS17x, Trimble). The driver reads out much of the internal GPS data and makes it accessible via clock variables. To find out about additional variable names, query for the clock_var_list variable on a specific clock association as shown below.

First let ntpq display the table of associations:

```
ntpq> as ind assID status conf reach auth condition last_event cnt
```

Then switch to raw output. This may be required because of display limitations in ntpq/ntpd - so large lists need to be retrieved in several queries.

```
ntpq> raw  Output set to raw
```

Use the cv command to read the list of clock variables of a selected association:

```
ntpq> cv 19557 clock_var_list
```

The long output of the command above looks similar to:

```
assID=19557 status=0x0000, clock_var_list="type,timecode,poll,noreply,badformat,baddata,fudgetime1, ... gps_health[3],gps_cfg[4],gps_health[4],gps_cfg[5]"
```

Then use the cv command again to list selected clock variables. The following command must be entered as a single line:

```
ntpq> cv 19557 refclock_status,refclock_format,refclock_states,refclock_id, ... gps_health[1],gps_cfg[2],gps_health[2],gps_cfg[3],gps_health[3],gps_cfg[4],gps_health[4],gps_cfg[5]
```

The output of the command above is wrapped around depending on the screen width and looks similar to:

```
```

10.7.8 Fudge Factors

**time1 time**  Specifies the time offset calibration factor, in seconds and fraction. The default value depends on the clock type.

**time2 time**  If flag1 is 0, time2 specifies the offset of the PPS signal from the actual time (PPS fine tuning). If flag1 is 1, time2 specifies the number of seconds a receiver with a premium local oscillator can be trusted after losing synchronisation.

**stratum stratum**  The stratum for this reference clock.

**refid refid**  The refid for this reference clock.

**flag1 { 0 1 1 }**  If 0, the fudge factor time2 refers to the PPS offset. If 1, time2 refers to the TRUST TIME.
flag2 { 0 1 1 } If flag2 is 1, sample PPS on CLEAR instead of on ASSERT.
flag3 { 0 1 1 } If flag3 is 1, link kernel PPS tracking to this refclock instance.
flag4 { 0 1 1 } Delete next leap second instead of adding it. (You’ll need to wait a bit for that to happen 8-)

Note about auxiliary Sun STREAMS modules (SunOS and Solaris):
The timecode of these receivers can be sampled via a STREAMS module in the kernel. (The STREAMS module has been designed for use with Sun systems under SunOS 4.1.x or Solaris 2.3 - 2.8. It can be linked directly into the kernel or loaded via the loadable driver mechanism.) This STREAMS module can be adapted to convert different time code formats. Nowadays the PPSAPI mechanism is usually used.

10.7.9 Making your own PARSE clocks

The parse clock mechanism deviates from the way other NTP reference clocks work. For a short description of how to build parse reference clocks, see making PARSE clocks.

10.7.10 Additional Information

Reference Clock Drivers

10.8 Magnavox MX4200 GPS Receiver

10.8.1 Synopsis

Address: 127.127.9.u
Reference ID: GPS
Driver ID: GPS_MX4200
Serial Port: /dev/gpsu; 4800 baud, 8-bits, no parity
Features: ppsclock (required)

10.8.2 Description

This driver supports the Magnavox MX4200 Navigation Receiver adapted to precision timing applications. This driver supports all compatible receivers such as the 6-channel MX4200, MX4200D, and the 12-channel MX9212, MX9012R, MX9112.
Leica MX9400N Navigator.

Leica Geosystems acquired the Magnavox commercial GPS technology business in February of 1994. They now market and support former Magnavox GPS products such as the MX4200 and its successors.

### 10.8.3 Operating Modes

This driver supports two modes of operation, static and mobile, controlled by clock flag 2.

In static mode (the default) the driver assumes that the GPS antenna is in a fixed location. The receiver is initially placed in a “Static, 3D Nav” mode, where latitude, longitude, elevation and time are calculated for a fixed station. An average position is calculated from this data. After 24 hours, the receiver is placed into a “Known Position” mode, initialized with the calculated position, and then solves only for time.

In mobile mode, the driver assumes the GPS antenna is mounted on a moving platform such as a car, ship, or aircraft. The receiver is placed in “Dynamic, 3D Nav” mode and solves for position, altitude and time while moving. No position averaging is performed.

### 10.8.4 Monitor Data

The driver writes each timecode as received to the `clockstats` file. Documentation for the NMEA-0183 proprietary sentences produced by the MX4200 can be found in [MX4200 Receiver Data Format](#).

### 10.8.5 Fudge Factors

- **time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
- **time2 time** Not used by this driver.
- **stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- **refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.
- **flag1 0 | 1** Not used by this driver.
- **flag2 0 | 1** Assume GPS receiver is on a mobile platform if set.
- **flag3 0 | 1** Not used by this driver.
- **flag4 0 | 1** Not used by this driver.

### 10.8.6 Additional Information

Reference Clock Drivers

### 10.9 Austron 2200A/2201A GPS Receivers

#### 10.9.1 Synopsis

Address: 127.127.10.u
Reference ID: GPS
Driver ID: GPS_AS2201
Serial Port: /dev/gpsu; 9600 baud, 8-bits, no parity
Features: tty_clk

10.9.2 Description

This driver supports the Austron 2200A/2201A GPS/LORAN Synchronized Clock and Timing Receiver connected via a serial port. It supports several special features of the clock, including the Input Buffer Module, Output Buffer Module, IRIG-B Interface Module and LORAN Assist Module. It requires the RS232 Buffered Serial Interface module for communication with the driver.

For use with a single computer, the receiver can be connected directly to the receiver. For use with multiple computers, one of them is connected directly to the receiver and generates the polling messages. The other computers just listen to the receiver output directly or through a buffer amplifier. For computers that just listen, fudge flag2 must be set and the ppsclock streams module configured on each of them.

This receiver is capable of a comprehensive and large volume of statistics and operational data. The specific data collection commands and attributes are embedded in the driver source code; however, the collection process can be enabled or disabled using the flag4 flag. If set, collection is enabled; if not, which is the default, it is disabled. A comprehensive suite of data reduction and summary scripts is in the ./scripts/stats directory of the ntp3 distribution.

10.9.3 Monitor Data

When enabled by the flag4 fudge flag, every received timecode is written as-is to the clockstats file.

10.9.4 Fudge Factors

**time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2 time** Not used by this driver.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.

**flag1 0 | 1** Not used by this driver.

**flag2 0 | 1** Set for computers that listen-only.

**flag3 0 | 1** Not used by this driver.

**flag4 0 | 1** Enable verbose clockstats recording if set.

10.9.5 Additional Information

Reference Clock Drivers

10.10 Arbiter 1088A/B GPS Receiver

10.10.1 Synopsis

Address: 127.127.11.u
Reference ID: GPS
Driver ID: GPS_ARBITER
Serial Port: /dev/gpsu; 9600 baud, 8-bits, no parity
Features: tty_clk

10.10.2 Description

This driver supports the Arbiter 1088A/B Satellite Controlled Clock. The claimed accuracy of this clock is 100 ns relative to the PPS output when receiving four or more satellites.

The receiver should be configured before starting the NTP daemon, in order to establish reliable position and operating conditions. It does not initiate surveying or hold mode. For use with NTP, the daylight savings time feature should be disabled (D0 command) and the broadcast mode set to operate in UTC (BU command).

The timecode format supported by this driver is selected by the poll sequence B5, which initiates a line in the following format to be repeated once per second until turned off by the B0 command.

Format B5 (24 ASCII printing characters):

```
<cr><lf>i yy ddd hh:mm:ss.000bbb
```

<table>
<thead>
<tr>
<th>i</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>clock locked, maximum accuracy</td>
</tr>
<tr>
<td>'?'</td>
<td>clock failure, time not reliable</td>
</tr>
<tr>
<td>4</td>
<td>clock unlocked, accuracy &lt; 1 us</td>
</tr>
<tr>
<td>5</td>
<td>clock unlocked, accuracy &lt; 10 us</td>
</tr>
<tr>
<td>6</td>
<td>clock unlocked, accuracy &lt; 100 us</td>
</tr>
<tr>
<td>7</td>
<td>clock unlocked, accuracy &lt; 1 ms</td>
</tr>
<tr>
<td>8</td>
<td>clock unlocked, accuracy &lt; 10 ms</td>
</tr>
<tr>
<td>9</td>
<td>clock unlocked, accuracy &lt; 100 ms</td>
</tr>
<tr>
<td>A</td>
<td>clock unlocked, accuracy &lt; 1 s</td>
</tr>
<tr>
<td>B</td>
<td>clock unlocked, accuracy &lt; 10 s</td>
</tr>
</tbody>
</table>

The status string is encoded as follows:

Format SR (25 ASCII printing characters)

```
V=vv S=ss T=t P=pdop E=ee
```

<table>
<thead>
<tr>
<th>vv</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>satellites visible</td>
</tr>
<tr>
<td>ss</td>
<td>relative signal strength</td>
</tr>
<tr>
<td>t</td>
<td>satellites tracked</td>
</tr>
<tr>
<td>pdop</td>
<td>position dilution of precision (meters)</td>
</tr>
<tr>
<td>ee</td>
<td>hardware errors</td>
</tr>
</tbody>
</table>
A three-stage median filter is used to reduce jitter and provide a dispersion measure. The driver makes no attempt to correct for the intrinsic jitter of the radio itself.

10.10.3 Monitor Data

When enabled by the flag4 fudge flag, an additional line containing the latitude, longitude, elevation and optional deviation data is written to the clockstats file. The deviation data operates with an external pulse-per-second (PPS) input, such as a cesium oscillator or another radio clock. The PPS input should be connected to the B event channel and the radio initialized for deviation data on that channel. The deviation data consists of the mean offset and standard deviation of the external PPS signal relative the GPS signal, both in microseconds over the last 16 seconds.

10.10.4 Fudge Factors

_time1 time_ Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
_time2 time_ Not used by this driver.
_stratum number_ Specifies the driver stratum, in decimal from 0 to 15, with default 0.
_refid string_ Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.
_flag1 0 | 1_ Not used by this driver.
_flag2 0 | 1_ Not used by this driver.
_flag3 0 | 1_ Not used by this driver.
_flag4 0 | 1_ Enable verbose clockstats recording if set.

10.10.5 Additional Information

Reference Clock Drivers

10.11 KSI/Odetics TPRO/S IRIG Interface

10.11.1 Synopsis

Address: 127.127.12.12
Reference ID: IRIG
Driver ID: IRIG_TPRO
TPRO Device: /dev/tprou
Requires: KSI/Odetics device driver, /usr/include/sys/tpro.h header file

10.11.2 Description

This driver supports the KSI/Odetics TPRO and TPRO-SAT IRIG-B Decoder, which is a module connected directly to the SBus of a Sun workstation. The module works with the IRIG-B signal generated by several radio clocks, including those made by Arbiter, Austron, Odetics, Spectracom and TrueTime, among others, although it is generally an add-on option. In the case of the TPRO-SAT, the module is an integral part of a GPS receiver, which serves as the primary timing source.
Using the TPRO interface as a NTP reference clock provides precision time only to ntpd and its clients. With suitable kernel modifications, it is possible to use the TPRO as the CPU system clock, avoiding errors introduced by the CPU clock oscillator wander. See the A Kernel Model for Precision Timekeeping page for further details.

### 10.11.3 Fudge Factors

- **time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
- **time2 time** Not used by this driver.
- **stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- **refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default IRIG.
- **flag1 0 | 1** Not used by this driver.
- **flag2 0 | 1** Not used by this driver.
- **flag3 0 | 1** Not used by this driver.
- **flag4 0 | 1** Not used by this driver.

### 10.11.4 Additional Information

Reference Clock Drivers

### 10.12 bc635VME/bc350VXI Time and Frequency Processor

#### 10.12.1 Synopsis

Address: 127.127.16.\textit{u}
Reference ID: BTfp
Driver ID: GPS_BANCOMM
Bancomm Device \texttt{/dev/btfp0}
Requires: Bancomm bc635 TFP device module driver for SunOS 4.x/SunOS 5.x

#### 10.12.2 Description

This is the clock driver for the Bancomm bc635VME Time and Frequency Processor. It requires the BANCOMM bc635VME bc350VXI Time and Frequency Processor Module Driver for SunOS 4.x/SunOS 5.x UNIX Systems.

Most of this code is originally from refclock_bancomm.c with thanks. It has been modified and tested on an UltraSparc Iii-cEngine running Solaris 2.6. A port for HPUX is not available henceforth.

#### 10.12.3 Additional Information

Reference Clock Drivers
10.13 NIST/USNO/PTB Modem Time Services

10.13.1 Synopsis

Address: 127.127.18.u
Reference ID: NIST | USNO | PTB | WWVB
Driver ID: ACTS_MODEM
Serial Port: /dev/actsu; 9600 baud, 8-bits, no parity
Features: tty_clk
Requires: /usr/include/sys/termios.h header file with modem control and a dial-out (cua) device.

10.13.2 Description

This driver supports the US (NIST and USNO) and European (PTB (Germany), NPL (UK), etc.) modem time services, as well as Spectracom GPS and WWVB receivers connected via a modem. The driver periodically dials a number from a telephone list, receives the timecode data and calculates the local clock correction. It is designed primarily for backup when neither a radio clock nor connectivity to Internet time servers are available. It can also be configured to operate full period.

For best results the indicated time must be corrected for the modem and telephone circuit propagation delays, which can reach 200 ms or more. For the NIST service, corrections are determined automatically by measuring the roundtrip delay of echoed characters. With this service the absolute accuracy is typically a millisecond or two. Corrections for the other services must be determined by other means. With these services variations from call to call and between messages during a call are typically a few milliseconds, occasionally higher.

This driver requires a 9600-bps modem with a Hayes-compatible command set and control over the modem data terminal ready (DTR) control line. The actual line speed ranges from 1200 bps with USNO to 14,400 bps with NIST. The modem setup string is hard-coded in the driver and may require changes for nonstandard modems or special circumstances.

There are three modes of operation selected by the mode keyword in the server configuration command. In manual mode (2) the calling program is initiated by setting fudge flag1. This can be done manually using ntpq, or by a cron job. In auto mode (0) flag1 is set at each poll event. In backup mode (1) flag1 is set at each poll event, but only if no other synchronization sources are available.

When flag1 is set, the calling program dials the first number in the list specified by the phone command. If the call fails for any reason, the program dials the second number and so on. The phone number is specified by the Hayes ATDT prefix followed by the number itself, including the prefix and long-distance digits and delay code, if necessary. The flag1 is reset and the calling program terminated if (a) valid clock update has been determined, (b) no more numbers remain in the list, (c) a device fault or timeout occurs or (d) fudge flag1 is reset manually using ntpq.

The driver automatically recognizes the message format of each modem time service. It selects the parsing algorithm depending on the message length. There is some hazard should the message be corrupted. However, the data format is checked carefully and only if all checks succeed is the message accepted. Corrupted lines are discarded without complaint. Once the service is known, the reference identifier for the driver is set to NIST, USNO, PTB or WWVB as appropriate.

The Spectracom radio can be connected via a modem if the radio is configured to send time codes continuously at 1-s intervals. In principle, fudge flag2 enables port locking, allowing the modem to be shared when not in use by this driver. At least on Solaris with the current NTP I/O routines, this results in lots of ugly error messages.

The minpoll and maxpoll keywords of the server configuration command can be used to limit the intervals between calls. The recommended settings are 12 (1.1 hours) for minpoll and 17 (36 hours) for maxpoll. Ordinarily, the poll interval will start at minpoll and ramp up to maxpoll in a day or two.
10.13.3 US Phone Numbers and Formats

Note: Phone numbers include the entire Hayes modem command, including the ATDT and other control codes as may be necessary. For most cases only the ATDT may be necessary.

National Institute of Science and Technology (NIST)
Phone: (303) 494-4774 (Boulder, CO); (808) 335-4721 (Hawaii)

Data Format

<table>
<thead>
<tr>
<th>National Institute of Standards and Technology</th>
<th>Telephone Time Service, Generator 3B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter question mark &quot;?&quot; for HELP</td>
<td></td>
</tr>
<tr>
<td>MJD YR MO DA H M S ST S UT1 msADV &lt;OTM&gt;</td>
<td></td>
</tr>
<tr>
<td>47999 90-04-18 21:39:15 50 0 +.1 045.0 UTC(NIST) *</td>
<td></td>
</tr>
<tr>
<td>47999 90-04-18 21:39:16 50 0 +.1 045.0 UTC(NIST) #</td>
<td></td>
</tr>
</tbody>
</table>

MJD, YR, ST, UT1 and UTC(NIST) are not used by this driver. The <OTM> on-time character “*” changes to “#” when the delay correction is valid.

US Naval Observatory (USNO)
Phone: (202) 762-1594 (Washington, DC); (719) 567-6742 (Boulder, CO)

Data Format (two lines, repeating at one-second intervals)

<table>
<thead>
<tr>
<th>jjjjjj</th>
</tr>
</thead>
<tbody>
<tr>
<td>nnn hhmss UTC</td>
</tr>
</tbody>
</table>

\* on-time character for previous timecode message
jjjjj modified Julian day number (not used)
nnn day of year
hhmss second of day

European Phone Numbers and Formats

Spectracom GPS and WWVB Receivers

If a modem is connected to a Spectracom receiver, this driver will call it and retrieve the time in one of two formats, 0 and 2. Ordinarily, the receiver requires a T in order to return the timecode. As this driver does not send data via the modem, it must either be configured in continuous mode or be polled by another local driver.

10.13.4 Monitor Data

The received timecode is written as-is to the clockstats file along with the Hayes connection and hang-up commands and result codes.

10.13.5 Fudge Factors

**time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2 time** Not used by this driver.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Set by the driver to (one of) NIST, USNO, PTB or WWVB.

**flag1 0 | 1** Initiate a call if 1. Automatically reset by program.

**flag2 0 | 1** Enables port locking if 1, disables if 0 (default).
10.14 Heath WWV/WWVH Receiver

10.14.1 Synopsis

Address: 127.127.19.\texttt{u}
Reference ID: \texttt{WWV}
Driver ID: \texttt{WWV\_HEATH}
Serial Port: /dev/heath\texttt{u}; 1200 baud, 8-bits, no parity
Features: tty\_clk
Requires: /usr/include/sys/termios.h header file with modem control

10.14.2 Description

This driver supports the Heath GC-1000 Most Accurate Clock, with RS232C Output Accessory. This is a WWV/WWVH receiver somewhat less robust than other supported receivers. Its claimed accuracy is 100 ms when actually synchronized to the broadcast signal, but this doesn’t happen even most of the time, due to propagation conditions, ambient noise sources, etc. When not synchronized, the accuracy is at the whim of the internal clock oscillator, which can wander into the sunset without warning. Since the indicated precision is 100 ms, expect a host synchronized only to this thing to wander to and fro, occasionally being rudely stepped when the offset exceeds the default CLOCK\_MAX of 128 ms.

The internal DIP switches should be set to operate at 1200 baud in MANUAL mode and the current year. The external DIP switches should be set to GMT and 24-hour format. It is very important that the year be set correctly in the DIP switches; otherwise, the day of year will be incorrect after 28 April of a normal or leap year.

In MANUAL mode the clock responds to a rising edge of the request to send (RTS) modem control line by sending the timecode. Therefore, it is necessary that the operating system implement the \texttt{TIOCMBIC} and \texttt{TIOCMBIS} ioctl system calls and \texttt{TIOCM\_RTS} control bit. Present restrictions require the use of a POSIX-compatible programming interface, although other interfaces may work as well.

The clock message consists of 23 ASCII printing characters in the following format:

\texttt{hh:mm:ss.f dd/mm/yr<cr>}

\texttt{hh:mm:ss.f} = hours, minutes, seconds
\texttt{f} = deciseconds (‘?’ when out of spec)
\texttt{dd/mm/yr} = day, month, year

The alarm condition is indicated by ‘?’, rather than a digit, at A. Note that 0?:?:?:?: is displayed before synchronization is first established and \texttt{hh:mm:ss.?} once synchronization is established and then lost again for about a day.

A fudge time1 value of .07 s appears to center the clock offset residuals.
10.14.3 Fudge Factors

`time1 time` Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

`time2 time` Not used by this driver.

`stratum number` Specifies the driver stratum, in decimal from 0 to 15, with default 0.

`refid string` Specifies the driver reference identifier, an ASCII string from one to four characters, with default WWV.

`flag1 0 | 1` Not used by this driver.

`flag2 0 | 1` Not used by this driver.

`flag3 0 | 1` Not used by this driver.

`flag4 0 | 1` Not used by this driver

Additional Information

Reference Clock Drivers

10.15 Generic NMEA GPS Receiver

10.15.1 Synopsis

Address: 127.127.20.u
Reference ID: GPS
Driver ID: GPS_NMEA
Serial Port: /dev/gpsu; 4800 - 115200 bps, 8-bits, no parity
Serial Port: /dev/gpsppsu; for just the PPS signal (this is tried first for PPS, before /dev/gpsu)
Serial Port: /dev/gpsu; symlink to server:port (for nmead)
Features: tty_clk

10.15.2 Description

This driver supports GPS receivers with the `$GPRMC`, `$GPGLL`, `$GPGGA`, `$GPZDA` and `$GPZDG` NMEA sentences by default. Note that Accord’s custom NMEA sentence `$GPZDG` reports using the GPS timescale, while the rest of the sentences report UTC. The difference between the two is a whole number of seconds which increases with each leap second insertion in UTC. To avoid problems mixing UTC and GPS timescales, the driver disables processing of UTC sentences once `$GPZDG` is received.

The driver expects the receiver to be set up to transmit at least one supported sentence every second.

The accuracy depends on the receiver used. Inexpensive GPS models are available with a claimed PPS signal accuracy of 1 \(\mu s\) or better relative to the broadcast signal. However, in most cases the actual accuracy is limited by the precision of the timecode and the latencies of the serial interface and operating system.

If the Operating System supports PPSAPI (RFC 2783), fudge flag1 1 enables its use.

The various GPS sentences that this driver recognises look like this:

(others quietly ignored)
Table: Accepted NMEA sentences

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning and Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTC</td>
<td>Time of day on UTC timescale. Hours, minutes and seconds [fraction (opt.)]. (hhmmss[.fff])</td>
</tr>
<tr>
<td>POS_STAT</td>
<td>Position status. (A = Data valid, V = Data invalid)</td>
</tr>
<tr>
<td>LAT</td>
<td>Latitude (llll.ll)</td>
</tr>
<tr>
<td>LAT_REF</td>
<td>Latitude direction. (N = North, S = South)</td>
</tr>
<tr>
<td>LON</td>
<td>Longitude (yyyyy.yy)</td>
</tr>
<tr>
<td>LON_REF</td>
<td>Longitude direction (E = East, W = West)</td>
</tr>
<tr>
<td>SPD</td>
<td>Speed over ground. (knots) (x.x)</td>
</tr>
<tr>
<td>HDG</td>
<td>Heading/track made good (degrees True) (x,x)</td>
</tr>
<tr>
<td>DATE</td>
<td>Date (ddmmyy)</td>
</tr>
<tr>
<td>MAG_VAR</td>
<td>Magnetic variation (degrees) (x.x)</td>
</tr>
<tr>
<td>MAG_REF</td>
<td>Magnetic variation (E = East, W = West)</td>
</tr>
<tr>
<td>FIX_MODE</td>
<td>Position Fix Mode (0 = Invalid, &gt;0 = Valid)</td>
</tr>
<tr>
<td>SAT_USED</td>
<td>Number of Satellites used in solution</td>
</tr>
<tr>
<td>HDOP</td>
<td>Horizontal Dilution of Precision</td>
</tr>
<tr>
<td>ALT</td>
<td>Antenna Altitude</td>
</tr>
<tr>
<td>ALT_UNIT</td>
<td>Altitude Units (Metres/Feet)</td>
</tr>
<tr>
<td>GEO</td>
<td>Geoid/Ellipsoid separation</td>
</tr>
<tr>
<td>G_UNIT</td>
<td>Geoid units (M/F)</td>
</tr>
<tr>
<td>D_AGE</td>
<td>Age of last DGPS Fix</td>
</tr>
<tr>
<td>D_REF</td>
<td>Reference ID of DGPS station</td>
</tr>
<tr>
<td>GPSTIME</td>
<td>Time of day on GPS timescale. Hours, minutes and seconds [fraction (opt.)]. (hhmmss[.f])</td>
</tr>
<tr>
<td>DD</td>
<td>Day of the month (1-31)</td>
</tr>
<tr>
<td>MM</td>
<td>Month of the year (1-12)</td>
</tr>
<tr>
<td>YYYY</td>
<td>Year</td>
</tr>
<tr>
<td>AA.BB</td>
<td>Denotes the signal strength (should be &lt; 05.00)</td>
</tr>
<tr>
<td>V</td>
<td><strong>GPS sync status</strong>  ‘0’ =&gt; INVALID time,</td>
</tr>
<tr>
<td></td>
<td>‘1’ =&gt; accuracy of +/- 20ms,</td>
</tr>
<tr>
<td></td>
<td>‘2’ =&gt; accuracy of +/- 100ns</td>
</tr>
<tr>
<td>CS</td>
<td>Checksum</td>
</tr>
<tr>
<td>&lt;cr&gt;&lt;lf&gt;</td>
<td>Sentence terminator.</td>
</tr>
</tbody>
</table>

Table: NMEA data items

10.15.3 The ‘mode’ byte

Specific GPS sentences and bitrates may be selected by setting bits of the ‘mode’ in the server configuration line:
server 127.127.20.x mode X

<table>
<thead>
<tr>
<th>Bit</th>
<th>Decimal</th>
<th>Hex</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>process $GPMRC</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>process $GPGGA</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>process $GPGLL</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>process $GPZDA or $GPZDG</td>
</tr>
<tr>
<td>4- 6</td>
<td></td>
<td></td>
<td>linespeed 4800 bps</td>
</tr>
<tr>
<td>16</td>
<td>0x10</td>
<td></td>
<td>linespeed 9600 bps</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td></td>
<td>linespeed 19200 bps</td>
</tr>
<tr>
<td>48</td>
<td>0x30</td>
<td></td>
<td>linespeed 38400 bps</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td></td>
<td>linespeed 57600 bps</td>
</tr>
<tr>
<td>80</td>
<td>0x50</td>
<td></td>
<td>linespeed 115200 bps</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>0x80</td>
<td>Write the sub-second fraction of the receive time stamp to the clockstat file for all recognised NMEA sentences. This can be used to get a useful value for fudge time2. <strong>Caveat:</strong> This will fill your clockstat file rather fast. Use it only temporarily to get the numbers for the NMEA sentence of your choice.</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>0x100</td>
<td>process $PGRMF</td>
</tr>
<tr>
<td>9- 15</td>
<td>0xFE0</td>
<td></td>
<td>reserved - leave 0</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>0x10000</td>
<td>Append extra statistics to the clockstats line. Details below.</td>
</tr>
</tbody>
</table>

Table: mode byte bits and bit groups

The default (mode 0) is to process all supported sentences at a linespeed of 4800 bps, which results in the first one received and recognised in each cycle being used. If only specific sentences should be recognised, then the mode byte must be chosen to enable only the selected ones. Multiple sentences may be selected by adding their mode bit values, but of those enabled still only the first received sentence in a cycle will be used. Using more than one sentence per cycle is impossible, because

- there is only *fudge time2* available to compensate for transmission delays but every sentence would need a different one and
- using more than one sentence per cycle overstuff the internal data filters.

The driver uses 4800 bits per second by default, but faster bitrates can be selected using bits 4 to 6 of the mode field. **Caveat:** Using higher line speeds does not necessarily increase the precision of the timing device. Higher line speeds are not necessarily helpful for the NMEA driver, either. They can be used to accomodate for an amount of data that does not fit into a 1-second cycle at 4800 bps, but high-speed high-volume NMEA data is likely to cause trouble with the serial line driver since NMEA supports no protocol handshake. Any device that is exclusively used for time synchronisation purposes should be configured to transmit the relevant data only, e.g. one $GPRMC or $GPZDA per second, at a linespeed of 4800 bps or 9600 bps.

### 10.15.4 Monitor Data

The last GPS sentence that is accepted or rejected is written to the clockstats file and available with `ntpq -c clockvar`. (Logging the rejected sentences lets you see/debug why they were rejected.) Filtered sentences are not logged.

If the 0x10000 mode bit is on and clockstats is enabled, several extra counters will be appended to the NMEA sentence that gets logged. For example:
Table: Clockstats

Sentences like $GPGSV that don’t contain the time will get counted in the total but otherwise ignored.

Configuring NMEA Refclocks might give further useful hints for specific hardware devices that exhibit strange or curious behaviour.

To make a specific setting, select the corresponding decimal values from the mode byte table, add them all together and enter the resulting decimal value into the clock configuration line.

### 10.15.5 Setting up the Garmin GPS-25XL

Switch off all output with by sending it the following string.

```
"$PGRMO,,2<cr><lf>"
```

Now switch only $GPRMC on by sending it the following string.

```
"$PGRMO,GPRMC,1<cr><lf>"
```

On some systems the PPS signal isn’t switched on by default. It can be switched on by sending the following string.

```
"$PGRMC,,,,,,,,,,,,2<cr><lf>"
```

### 10.15.6 Fudge Factors

- **time1 time** Specifies the PPS time offset calibration factor, in seconds and fraction, with default 0.0.
- **time2 time** Specifies the serial end of line time offset calibration factor, in seconds and fraction, with default 0.0.
- **stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.
- **refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.
- **flag1 0 | 1** Disable PPS signal processing if 0 (default); enable PPS signal processing if 1.
- **flag2 0 | 1** If PPS signal processing is enabled, capture the pulse on the rising edge if 0 (default); capture on the falling edge if 1.
- **flag3 0 | 1** If PPS signal processing is enabled, use the ntpd clock discipline if 0 (default); use the kernel discipline if 1.
- **flag4 0 | 1** Obscures location in timecode: 0 for disable (default), 1 for enable.
Additional Information

flag1, flag2, and flag3 are ignored under Windows.

Reference Clock Drivers

10.16 PPS Clock Discipline

10.16.1 Synopsis

Address: 127.127.22\u
Reference ID: PPS
Driver ID: PPS
Serial or Parallel Port: /dev/pps\u
Requires: PPSAPI signal interface for PPS signal processing.

Note: This driver supersedes an older one of the same name. The older driver operated with several somewhat archaic signal interface devices, required intricate configuration and was poorly documented. This driver requires the Pulse per Second API (PPSAPI):sup:1. Note also that the pps configuration command has been obsoleted by this driver.

10.16.2 Description

This driver furnishes an interface for the pulse-per-second (PPS) signal produced by a cesium clock, radio clock or related devices. It can be used to augment the serial timecode generated by a GPS receiver, for example. It can be used to remove accumulated jitter and re-time a secondary server when synchronized to a primary server over a congested, wide-area network and before redistributing the time to local clients. The driver includes extensive signal sanity checks and grooming algorithms. A range gate and frequency discriminator reject noise and signals with incorrect frequency. A multiple-stage median filter rejects jitter due to hardware interrupt and operating system latencies. A trimmed-mean algorithm determines the best time samples. With typical workstations and processing loads, the incidental jitter can be reduced to a few microseconds.

While this driver can discipline the time and frequency relative to the PPS source, it cannot number the seconds. For this purpose an auxiliary source is required, ordinarily a radio clock operated as a primary reference (stratum 1) source; however, another NTP time server can be used as well. For this purpose, the auxiliary source should be specified as the prefer peer, as described in the Mitigation Rules and the prefer Keyword page.

The driver requires the PPSAPI interface 1, which is a proposed IETF standard. The interface consists of the timepps\u.h header file and associated kernel support. Support for this interface is included in current versions of Solaris, FreeBSD and Linux and proprietary versions of Tru64 (Alpha) and SunOS. See the Pulse-per-second (PPS) Signal Interfacing page for further information.

The PPS source can be connected via a serial or parallel port, depending on the hardware and operating system. A serial port can be dedicated to the PPS source or shared with another device; however, if dedicated the data leads should not be connected, as noise or unexpected signals can cause ntpd to exit.

A radio clock is usually connected via a serial port and the PPS source connected via a level converter to the data carrier detect (DCD) pin (DB-9 pin 1, DB-25 pin 8) of the same connector. In some systems where a parallel port and driver are available, the PPS signal can be connected directly to the ACK pin (DB25 pin 10) of the connector. Whether the PPS signal is connected via a dedicated port or shared with another device, the driver opens the device

---

/dev/pps%d, where %d is the unit number. As with other drivers, links can be used to redirect the logical name to the actual physical device.

The driver normally operates like any other driver and uses the same mitigation algorithms and PLL/FLL clock discipline incorporated in the daemon. If kernel PLL/FLL support is available, the kernel PLL/FLL clock discipline can be used instead. The default behavior is not to use the kernel PPS clock discipline, even if present. This driver incorporates a good deal of signal processing to reduce jitter using the median filter algorithm in the driver. As the result, performance with minpoll configured at 4 (16s) is generally better than the kernel PPS discipline. However, fudge flag 3 can be used to enable the kernel PPS discipline if necessary.

This driver is enabled only under one of two conditions (a) a prefer peer other than this driver is among the survivors of the mitigation algorithms or (b) there are no survivors and the minsane option of the tos command is 0. The prefer peer designates another source that can reliably number the seconds when available. However, if no sources are available, the system clock continues to be disciplined by the PPS driver on an indefinite basis.

A scenario where the latter behavior can be most useful is a planetary orbiter fleet, for instance in the vicinity of Mars, where contact between orbiters and Earth only one or two times per Sol (Mars day). These orbiters have a precise timing reference based on an Ultra Stable Oscillator (USO) with accuracy in the order of a Cesium oscillator. A PPS signal is derived from the USO and can be disciplined from Earth on rare occasion or from another orbiter via NTP. In the above scenario the PPS signal disciplines the spacecraft clock between NTP updates.

In a similar scenario a PPS signal can be used to discipline the clock between updates produced by the modem driver. This would provide precise synchronization without needing the Internet at all.

### 10.16.3 Fudge Factors

**time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2 time** Not used by this driver.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default PPS.

**flag1 0 | 1** Not used by this driver.

**flag2 0 | 1** Specifies PPS capture on the rising (assert) pulse edge if 0 (default) or falling (clear) pulse edge if 1. Not used under Windows - if the special serialpps.sys serial port driver is installed then the leading edge will always be used.

**flag3 0 | 1** Controls the kernel PPS discipline: 0 for disable (default), 1 for enable. Not used under Windows - if the special serialpps.sys serial port driver is used then kernel PPS will be available and used.

**flag4 0 | 1** Record a timestamp once for each second if 1. Useful for constructing Allan deviation plots.

### 10.16.4 Additional Information

Reference Clock Drivers
10.17 Hewlett Packard 58503A GPS Receiver and HP Z3801A

10.17.1 Synopsis

Address: 127.127.26.u
Reference ID: GPS
Driver ID: GPS_HP
Serial Port: /dev/hpgpsu; 9600 baud, 8-bits, no parity, 19200 baud 7-bits, odd parity for the HP Z3801A

10.17.2 Description

This driver supports the HP 58503A Time and Frequency Reference Receiver and HP Z3801A GPS Receiver. They use HP SmartClock (TM) to implement an Enhanced GPS receiver. The receiver accuracy when locked to GPS in normal operation is better than 1 usec. The accuracy when operating in holdover is typically better than 10 us per day. It receiver should be operated with factory default settings. Initial driver operation: expects the receiver to be already locked to GPS, configured and able to output timecode format 2 messages.

The driver uses the poll sequence :PTIME:TCODE? to get a response from the receiver. The receiver responds with a timecode string of ASCII printing characters, followed by a <cr><lf>, followed by a prompt string issued by the receiver, in the following format:

T#yyyyymddhhmmssMFLRVcc<cr><lf>scpi >

The driver processes the response at the <cr> and <lf>, so what the driver sees is the prompt from the previous poll, followed by this timecode. The prompt from the current poll is (usually) left unread until the next poll. So (except on the very first poll) the driver sees this:

scpi >T#yyyyymddhhmmssMFLRVcc<cr><lf>

The T is the on-time character, at 980 msec. before the next 1PPS edge. The # is the timecode format type. We look for format 2. Without any of the CLK or PPS stuff, then, the receiver buffer timestamp at the <cr> is 24 characters later, which is about 25 msec. at 9600 bps, so the first approximation for fudge time1 is nominally -0.955 seconds. This number probably needs adjusting for each machine / OS type, so far: -0.955000 on an HP 9000 Model 712/80 HP-UX 9.05 -0.953175 on an HP 9000 Model 370 HP-UX 9.10

This driver will probably work with the 58503B and 59551A if they are setup appropriately.

To use an HP Z3801A, specify mode 1 on the server config line to setup the right line paramters.

The timekeeping portion of HP’s business has been sold to Symmetricom.

10.17.3 Fudge Factors

**time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2 time** Not used by this driver.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.

**flag1 0 | 1** Not used by this driver.
flag2 0 | 1 Not used by this driver.
flag3 0 | 1 Not used by this driver.
flag4 0 | 1 Not used by this driver.

10.18 Arcron MSF Receiver

10.18.1 Synopsis

Address: 127.127.27.u
Reference ID: MSFa / MSF / DCF / WWVB
Driver ID: MSF_ARCRON
Serial Port: /dev/arcu; 300 baud, 8-bits, 2-stop, no parity
Features: tty_clk

10.18.2 Description

This driver supports the Arcron MSF, DCF and WWVB receivers. The clock reports its ID as "MSFa", "MSF", "DCF" or "WWVB" to indicate the time source.

This documentation describes v1.3 (2003/2/21) of the source and has been tested against ntpd 4.1.0 on linux x86. Changes from v1.1 and v1.2 include patches to work with the new ntp-4 code, clock support for DCF and WWVB configurable via mode flag, an option to ignore resync request (for those of us at the fringes of the WWVB signal, for instance), averaging of the signal quality poll and several bug fixes, code cleanup and standardizations. In all other respects, the driver works as per v1.1 if a mode is not specified.

To use the alternate modes, the mode flag must be specified. If the mode flag is 0, or unspecified, the original MSF version is assumed. This should assure backwards compatibility and should not break existing setups.

The previous documentation described version V1.1 (1997/06/23) of the source and had been tested (amongst others) against ntpd3-5.90 on Solaris-1 (SunOS 4.1.3_U1 on an SS1 serving as a router and firewall) and against ntpd3-5.90 on Solaris-2.5 (on a SS1+ and TurboSPARC 170MHz). That code will claimed increased stability, reduced jitter and more efficiency (fewer context switches) with the tty_clk discipline/STREAMS module installed, but this has not been tested. For a to-do list see the comments at the start of the code.

This code has been significantly slimmed down since the V1.0 version, roughly halving the memory footprint of its code and data.

This driver is designed to allow the unit to run from batteries as designed, for something approaching the 2.5 years expected in the usual stand-alone mode, but no battery-life measurements have been taken.

Much of this code is originally from the other refclock driver files with thanks. The code was originally made to work with the clock by Derek Mulcahy, with modifications by Damon Hart-Davis. Thanks also to Lyndon David for some of the specifications of the clock. Paul Alfille added support for the WWVB clock. Christopher Price added enhanced support for the MSF, DCF and WWVB clocks.

There is support for a Tcl/Tk monitor written by Derek Mulcahy that examines the output stats; see the ARC Rugby MSF Receiver page for more details and the code. Information on the WWVB version is available from Atomic Time as their Atomic Time PC.

Look at the notes at the start of the code for further information; some of the more important details follow.

The driver interrogates the clock at each poll (ie every 64s by default) for a timestamp. The clock responds at the start of the next second (with the start bit of the first byte being on-time). In the default or original MSF mode, the time...
is in ‘local’ format, including the daylight savings adjustment when it is in effect. The driver code converts the time back to UTC. In modes 1-3 the driver can be configured for UTC or local time depending on the setting of flag1.

The clock claims to be accurate to within about 20ms of the broadcast time, and given the low data transmission speed from clock to host, and the fact that the clock is not in continuous sync with MSF, it seems sensible to set the ‘precision’ of this clock to -5 or -4, -4 being used in this code, which builds in a reported dispersion of over 63ms (ie says ‘This clock is not very good.’). You can improve the reported precision to -4 (and thus reduce the base dispersion to about 31ms) by setting the fudge flag3 to 1.

Even a busy and slow IP link can yield lower dispersions than this from polls of primary time servers on the Internet, which reinforces the idea that this clock should be used as a backup in case of problems with such an IP link, or in the unfortunate event of failure of more accurate sources such as GPS.

By default this clock reports itself to be at stratum 2 rather than the usual stratum 0 for a refclock, because it is not really suited to be used as other than a backup source. The stratum reported can be changed with the \texttt{stratum} directive to be whatever you like. After careful monitoring of your clock, and appropriate choice of the \texttt{time1} fudge factor to remove systematic errors in the clock’s reported time, you might fudge the clock to stratum 1 to allow a stratum-2 secondary server to sync to it.

In default mode, the driver code arranges to resync the clock to MSF at intervals of a little less than an hour (deliberately avoiding the same time each hour to avoid any systematic problems with the signal or host). Whilst resyncing, the driver supplements the normal polls for time from the clock with polls for the reception signal quality reported by the clock. If the signal quality is too low (0–2 out of a range of 0–5), we chose not to trust the clock until the next resync (which we bring forward by about half an hour). If we don’t catch the resync, and so don’t know the signal quality, we do trust the clock (because this would generally be when the signal is very good and a resync happens quickly), but we still bring the next resync forward and reduce the reported precision (and thus increase reported dispersion).

If we force resyncs to MSF too often we will needlessly exhaust the batteries the unit runs from. During clock resync this driver tries to take enough time samples to avoid \texttt{ntpd} losing sync in case this clock is the current peer. By default the clock would only resync to MSF about once per day, which would almost certainly not be acceptable for NTP purposes.

The driver does not force an immediate resync of the clock to MSF when it starts up to avoid excessive battery drain in case \texttt{ntpd} is going to be repeatedly restarted for any reason, and also to allow enough samples of the clock to be taken for \texttt{ntpd} to sync immediately to this clock (and not remain unsynchronised or to sync briefly to another configured peer, only to hop back in a few poll times, causing unnecessary disturbance). This behaviour should not cause problems because the driver will not accept the timestamps from the clock if the status flag delivered with the time code indicates that the last resync attempt was unsuccessful, so the initial timestamps will be close to reality, even if with up to a day’s clock drift in the worst case (the clock by default resyncs to MSF once per day).

When alternate modes 1-3 are selected, the driver can be configured to ignore the resync requests by setting flag2 to 1. This allows clocks at the fringe of the signal to resync at night when signals are stronger.

The clock has a peculiar RS232 arrangement where the transmit lines are powered from the receive lines, presumably to minimise battery drain. This arrangement has two consequences:

- Your RS232 interface must drive both +ve and -ve
- You must (in theory) wait for an echo and a further 10ms between characters

This driver, running on standard Sun and x86 hardware, seems to work fine; note the use of the \texttt{send\_slow()} routine to queue up command characters to be sent once every two seconds.

Three commands are sent to the clock by this driver. Each command consists of a single letter (of which only the bottom four bits are significant), followed by a CR (ASCII 13). Each character sent to the clock should be followed by a delay to allow the unit to echo the character, and then by a further 10ms. Following the echo of the command string, there may be a response (ie in the case of the g and o commands below), which in the case of the o command may be delayed by up to 1 second so as the start bit of the first byte of the response can arrive on time. The commands and their responses are:

\textbf{10.18. Arcron MSF Receiver}
**g CR** Request for signal quality. Answer only valid during (late part of) resync to MSF signal. The response consists of two characters as follows:

- **bit 7** parity
- **bit 6** always 0
- **bit 5** always 1
- **bit 4** always 1
- **bit 3** always 0
- **bit 2** always 0
- **bit 1** always 1
- **bit 0** = 0 if no reception attempt at the moment, = 1 if reception attempt (ie resync) in progress

**h CR** Request to resync to signal. Can take up from about 30s to 360s. Drains batteries so should not be used excessively. After this the clock time and date should be correct and the phase within 20ms of time as transmitted from the source signal (remember to allow for propagation time). By default the clock resyncs once per day in the late evening/early morning (presumably to catch transitions to/from daylight saving time quickly). This driver code, by default, resyncs at least once per hour to minimise clock wander.

**o CR** Request timestamp. Start bit of first byte of response is on-time, so may be delayed up to 1 second. Note that the driver will convert time to GMT, if required. The response data is as follows:

1. hours tens (hours range from 00 to 23)
2. hours units
3. minutes tens (minutes range from 00 to 59)
4. minutes units
5. seconds tens (seconds presumed to range from 00 to 60 to allow for leap second)
6. seconds units
7. day of week 1 (Monday) to 7 (Sunday)
8. day of month tens (day ranges from 01 to 31)
9. day of month units
10. month tens (months range from 01 to 12)
11. month units
12. year tens (years range from 00 to 99)
13. year units
14. BST/UTC status (Ignored in WWVB version)
bit 7  parity
bit 6  always 0
bit 5  always 1
bit 4  always 1
bit 3  (MSF) always 0
(WWVB) Leap year indicator bit
  0 = non-leap year
  1 = leap year
bit 2  = (MSF) 1 if UTC is in effect (reverse of bit 1)
(WWVB) Leap second warning bit
bit 1  = (MSF) 1 if BST is in effect (reverse of bit 2)
  = (WWVB) 0 if ST is in effect, 1 if DST is in effect, 1 if transition from ST with bit 0 is set to 0
bit 0  = (MSF) 1 if BST/UTC change pending
  = (WWVB) 0 if ST is in effect, 1 if DST is in effect, 0 if transition from DST with bit 1 is set to 0

15. clock status
bit 7  parity
bit 6  always 0
bit 5  always 1
bit 4  always 1
bit 3  = 1 if low battery is detected
bit 2  = 1 if last resync failed (though officially undefined for the MSF clock, officially defined for WWVB)
bit 1
  = 1 if at least one reception attempt was successful (MSF) since 0230 (DCF) since 0300 (WWVB)
  resets if not successful between 0300-0400
bit 0  = 1 if the clock has valid time—reset to zero when clock is reset (eg at power-up), and set to 1 after
  first successful resync attempt.

The driver only accepts time from the clock if the bottom three bits of the status byte are 011 or flag2 is set to 1 to ignore resync requests. For the MSF clock, if the UK parliament decides to move us to +0100/+0200 time as opposed to the current +0000/+0100 time, it is not clear what effect that will have on the time broadcast by MSF, and therefore on this driver’s usefulness.

A typical ntp.conf configuration file for this driver might be:

# hostname(n) means we expect (n) to be the stratum at which hostname runs.

# Default configuration (Original MSF mode)s...
skew 5000 mdev 0.05
server 127.127.27.0 mode 333

# ADJUST timel VALUE FOR YOUR HOST, CLOCK AND LOCATION!
NTP, Release 4.2.8p3

fudge 127.127.27.0 stratum
# WWVB users should change
server 127.127.27.0 mode 3
fudge 127.127.27.0 stratum

1 time1 0.016 flag3 1
that line to:
# ARCRON WWVB radio clock
1 time1 0.030 flag1 1 flag3 1

peer 11.22.33.9 # tick(1--2).
peer 11.22.33.4 # tock(3), boot/NFS server.
# This shouldn't get swept away unless left untouched for a long time.
driftfile /var/tmp/ntp.drift
#-----------------------------------------------------------------------------# RESTRICTIONS
# ============
# By default, don't trust and don't allow modifications.
restrict default ignore notrust nomodify

Ignore in fact.

# Allow others in our subnet to check us out...
restrict 11.22.33.0 mask 255.255.255.0 nomodify notrust
# Trust our peers for time. Don't trust others in case they are insane.
restrict 127.127.27.0 nomodify
restrict 11.22.33.4 nomodify
restrict 11.22.33.9 nomodify
# Allow anything from the local host.
restrict 127.0.0.1

There are a few #defines in the code that you might wish to play with:
ARCRON_KEEN With this defined, the code is relatively trusting of the clock, and assumes that you will have the
clock as one of a few time sources, so will bend over backwards to use the time from the clock when available
and avoid ntpd dropping sync from the clock where possible. You may wish to undefine this, especially if you
have better sources of time or your reception is ropey. However, there are many checks built in even with this
flag defined.
ARCRON_MULTIPLE_SAMPLES When is defined, we regard each character in the returned timecode as at a known
delay from the start of the second, and use the smallest (most negative) offset implied by any such character, ie
with the smallest kernel-induced display, and use that. This helps to reduce jitter and spikes.
ARCRON_LEAPSECOND_KEEN When is defined, we try to do a resync to MSF as soon as possible in the first hour
of the morning of the first day of the first and seventh months, ie just after a leap-second insertion or deletion
would happen if it is going to. This should help compensate for the fact that this clock does not continuously
sample MSF, which compounds the fact that MSF itself gives no warning of an impending leap-second event.
This code did not seem functional at the leap-second insertion of 30th June 1997 so is by default disabled.
PRECISION Currently set to -4, but you may wish to set it to -5 if you are more conservative, or to -6 if you have
particularly good experience with the clock and you live on the edge. Note that the flag3 fudge value will
improve the reported dispersion one notch if clock signal quality is known good. So maybe just leave this alone.

10.18.3 Monitor Data
Each timecode is written to the clockstats file with a signal quality value appended (‘0’–‘5’ as reported by the
clock, or ‘6’ for unknown).
Each resync and result (plus gaining or losing MSF sync) is logged to the system log at level LOG_NOTICE; note that
each resync drains the unit’s batteries, so the syslog entry seems justified.
212

Chapter 10. Reference Clock Drivers


Syslog entries are of the form:

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Process</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 10</td>
<td>10:15:24</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: unit 0: sending resync command</td>
</tr>
<tr>
<td>May 10</td>
<td>10:17:32</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: sync finished, signal quality 5: OK, will use clock</td>
</tr>
<tr>
<td>May 10</td>
<td>11:13:01</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: unit 0: sending resync command</td>
</tr>
<tr>
<td>May 10</td>
<td>11:14:06</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: sync finished, signal quality -1: UNKNOWN, will use clock</td>
</tr>
<tr>
<td>May 10</td>
<td>11:41:49</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: unit 0: sending resync command</td>
</tr>
<tr>
<td>May 10</td>
<td>11:43:57</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: sync finished, signal quality 5: OK, will use clock</td>
</tr>
<tr>
<td>May 10</td>
<td>12:39:26</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: unit 0: sending resync command</td>
</tr>
<tr>
<td>May 10</td>
<td>12:41:34</td>
<td>oolong</td>
<td>ntpd[615]: ARCRON: sync finished, signal quality 3: OK, will use clock</td>
</tr>
</tbody>
</table>

10.18.4 Fudge Factors

mode 0 | 1 | 2 | 3 Specifies the clock hardware model. This parameter is optional, it defaults to the original mode of operation. Supported modes of operation: 0 - Default, Original MSF 1 - Updated MSF 2 - New DCF77 3 - New WWVB

time1 time Specifies the time offset calibration factor, in seconds and fraction, with default 0.0. On a Sun SparcStation 1 running SunOS 4.1.3_U1, with the receiver in London, a value of 0.020 (20ms) seems to be appropriate.

time2 time Not currently used by this driver.

stratum number Specifies the driver stratum, in decimal from 0 to 15, with default 2. It is suggested that the clock be not be fudged higher than stratum 1 so that it is used a backup time source rather than a primary when more accurate sources are available.

refid string Specifies the driver reference identifier, an ASCII string from one to four characters, with default MSFa. When used in modes 1-3, the driver will report either MSF, DCF, or WWVB respectively.

flag1 0 | 1 (Modes 1-3) If set to 0 (the default), the clock is set to UTC time. If set to 1, the clock is set to localtime.

flag2 0 | 1 (Modes 1-3) If set to 0 (the default), the clock will be forced to resync approximately every hour. If set to 1, the clock will resync per normal operations (approximately midnight).

flag3 0 | 1 If set to 1, better precision is reported (and thus lower dispersion) while clock’s received signal quality is known to be good.

flag4 0 | 1 Not used by this driver.

10.18.5 Additional Information

Reference Clock Drivers
ARC Rugby MSF Receiver

10.19 Shared Memory Driver

10.19.1 Synopsis

Address: 127.127.28.u
Reference ID: SHM
Driver ID: SHM
10.19.2 Description

This driver receives its reference clock info from a shared memory-segment. The shared memory-segment is created with owner-only access by default, unless otherwise requested by the mode word for units ≥2. Units 0 and 1 are always created with owner-only access for backward compatibility.

10.19.3 Structure of shared memory-segment

```
struct shmTime {
    int mode; /* 0 - if valid is set:
               * use values,
               * clear valid
               * 1 - if valid is set:
               * if count before and after read of data is equal:
               * use values
               * clear valid */
    volatile int count;
    time_t clockTimeStampSec;
    int clockTimeStampUSec;
    time_t receiveTimeStampSec;
    int receiveTimeStampUSec;
    int leap;
    int precision;
    int nsamples;
    volatile int valid;
    unsigned clockTimeStampNSec; /* Unsigned ns timestamps */
    unsigned receiveTimeStampNSec; /* Unsigned ns timestamps */
    int dummy[8];
};
```

10.19.4 Operation mode=0

Each second, the value of valid of the shared memory-segment is checked:

If set, the values in the record (clockTimestampSec, clockTimestampUSec, receiveTimestampSec, receiveTimestampUSec, leap, precision) are passed to NTPD, and valid is cleared and count is bumped.

If not set, count is bumped.

10.19.5 Operation mode=1

Each second, valid in the shared memory-segment is checked:

If set, the count field of the record is remembered, and the values in the record (clockTimestampSec, clockTimestampUSec, receiveTimestampSec, receiveTimestampUSec, leap, precision) are read. Then, the remembered count is compared to current value of count now in the record. If both are equal, the values read from the record are passed to NTPD. If they differ, another process has modified the record while it was read out (was not able to produce this case), and failure is reported to NTPD. The valid flag is cleared and count is bumped.

If not set, count is bumped
10.19.6 Mode-independent post-processing

After the time stamps have been successfully plucked from the SHM segment, some sanity checks take place:

- The receive time stamp of the SHM data must be in the last 5 seconds before the time the data is processed. This helps in weeding out stale data.
- If the absolute difference between remote and local clock exceeds the limit (either time2 or the default of 4hrs), then the sample is discarded. This check is disabled when flag1 is set to 1.

10.19.7 GPSD

GPSD knows how to talk to many GPS devices. It can work with NTPD through the SHM driver.

The GPSD man page suggests setting minpoll and maxpoll to 4. That was an attempt to reduce jitter. The SHM driver was fixed (ntp-4.2.5p138) to collect data each second rather than once per polling interval so that suggestion is no longer reasonable.

Note: The GPSD client driver (type 46) uses the GPSD client protocol to connect and talk to GPSD, but using the SHM driver is the ancient way to have GPSD talk to NTPD. There are some tricky points when using the SHM interface to interface with GPSD, because GPSD will use two SHM clocks, one for the serial data stream and one for the PPS information when available. Receivers with a loose/sloppy timing between PPS and serial data can easily cause trouble here because NTPD has no way to join the two data streams and correlate the serial data with the PPS events.

10.19.8 Clockstats

If flag4 is set when the driver is polled, a clockstats record is written. The first 3 fields are the normal date, time, and IP address common to all clockstats records.

The 4th field is the number of second ticks since the last poll. The 5th field is the number of good data samples found. The last 64 will be used by NTPD. The 6th field is the number of sample that didn’t have valid data ready. The 7th field is the number of bad samples. The 8th field is the number of times the the mode 1 info was update while NTPD was trying to grab a sample.

Here is a sample showing the GPS reception fading out:

<table>
<thead>
<tr>
<th>54364</th>
<th>84927.157</th>
<th>127.127.28.0</th>
<th>66</th>
<th>66</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>54364</td>
<td>84990.161</td>
<td>127.127.28.0</td>
<td>63</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85053.160</td>
<td>127.127.28.0</td>
<td>63</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85116.159</td>
<td>127.127.28.0</td>
<td>63</td>
<td>62</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85180.158</td>
<td>127.127.28.0</td>
<td>64</td>
<td>63</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85246.161</td>
<td>127.127.28.0</td>
<td>66</td>
<td>66</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85312.157</td>
<td>127.127.28.0</td>
<td>66</td>
<td>50</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85375.160</td>
<td>127.127.28.0</td>
<td>63</td>
<td>41</td>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85439.155</td>
<td>127.127.28.0</td>
<td>64</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85505.158</td>
<td>127.127.28.0</td>
<td>66</td>
<td>36</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85569.157</td>
<td>127.127.28.0</td>
<td>64</td>
<td>0</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85635.157</td>
<td>127.127.28.0</td>
<td>66</td>
<td>0</td>
<td>66</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54364</td>
<td>85700.160</td>
<td>127.127.28.0</td>
<td>65</td>
<td>0</td>
<td>65</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

10.19.9 The ‘mode’ word

Some aspects of the driver behavior can be adjusted by setting bits of the ‘mode’ word in the server configuration line:

server 127.127.28.x mode Y

10.19. Shared Memory Driver
### Table: mode word bits and bit groups

<table>
<thead>
<tr>
<th>Bit</th>
<th>Dec</th>
<th>Hex</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>The SHM segment is private (mode 0600). This is the fixed default for clock units 0 and 1; clock units &gt;1 are mode 0666 unless this bit is set for the specific unit.</td>
</tr>
<tr>
<td>1-31</td>
<td></td>
<td></td>
<td>reserved – do not use</td>
</tr>
</tbody>
</table>

### 10.19.10 Fudge Factors

**time1 time** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2 time** Maximum allowed difference between remote and local clock, in seconds. Values 86400.0 are ignored, and the default value of 4hrs (14400s) is used instead. See also flag 1.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**reffid string** Specifies the driver reference identifier, an ASCII string from one to four characters, with default SHM.

**flag1 0 | 1** Skip the difference limit check if set. Useful for systems where the RTC backup cannot keep the time over long periods without power and the SHM clock must be able to force long-distance initial jumps. Check the difference limit if cleared (default).

**flag2 0 | 1** Not used by this driver.

**flag3 0 | 1** Not used by this driver.

**flag4 0 | 1** If flag4 is set, clockstats records will be written when the driver is polled.

### 10.19.11 Public vs. Private SHM segments

The driver attempts to create a shared memory segment with an identifier depending on the unit number. This identifier (which can be a numeric value or a string) clearly depends on the method used, which in turn depends on the host operating system:

- **Windows** uses a file mapping to the page file with the name ‘Global\NTPu’ for public accessible mappings, where \ is the clock unit. Private / non-public mappings are created as ‘Local\NTPu’.

  Public access assigns a NULL DACL to the memory mapping, while private access just uses the default DACL of the process creating the mapping.

- **SYSV IPC** creates a shared memory segment with a key value of 0x4E545030 + \, where \ is again the clock unit. (This value could be hex-decoded as ‘NTP0’, ‘NTP1’,..., with funny characters for units > 9.)

  Public access means a permission set of 0666, while private access creates the mapping with a permission set of 0600.

There’s no support for POSIX shared memory yet.

**NTPD** is started as root on most POSIX-like operating systems and uses the setuid/setgid system API to run under reduced rights once the initial setup of the process is done. One consequence out of this is that the allocation of SHM segments must be done early during the clock setup. The actual polling of the clock is done as the run-time user; deferring the creation of the SHM segment to this point will create a SHM segment owned by the runtime-user.
account. The internal structure of NTPD does not permit the use of a fudge flag if this is to be avoided; this is the reason why a mode bit is used for the configuration of a public segment.

When running under Windows, the chosen user account must be able to create a SHM segment in the global object name space for SHM clocks with public access. Otherwise the session isolation used by Windows kernels after WinXP will get into the way if the client program does not run in the same session.

10.19.12 Additional Information

Reference Clock Drivers

10.20 Trimble Palisade and Thunderbolt Receivers

10.20.1 Synopsis

<table>
<thead>
<tr>
<th>Address</th>
<th>127.127.29.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference ID:</td>
<td>GPS</td>
</tr>
<tr>
<td>Driver ID:</td>
<td>GPS_PALISADE</td>
</tr>
<tr>
<td>Serial Port:</td>
<td>/dev/palisade</td>
</tr>
<tr>
<td>Serial I/O:</td>
<td>9600 baud, 8-bits, 1-stop, odd parity</td>
</tr>
<tr>
<td>Serial I/O (Thunderbolt):</td>
<td>9600 baud, 8-bits, 1-stop, no parity</td>
</tr>
</tbody>
</table>
10.20.2 Description

The refclock_palisade driver supports Trimble Navigation’s Palisade Smart Antenna GPS receiver. Additional software and information about the Palisade GPS is available from: http://www.trimble.com/oem/ntp.


This documentation describes version 7.12 of the GPS Firmware and version 2.46 (July 15, 1999) and later, of the driver source.

This documentation describes version 1 of the Thunderbolt Receiver Firmware, no tests have been made on further firmwares, please read “Notes on the Thunderbolt Receiver’s Firmware” at the end of this documentation for more information.

10.20.3 Operating System Compatibility

The Palisade driver has been tested on the following software and hardware platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating System</th>
<th>NTP Sources</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386 (PC)</td>
<td>Linux</td>
<td>NTP Distribution</td>
<td>10 us</td>
</tr>
<tr>
<td>SUN</td>
<td>Solaris 2.x</td>
<td>NTP Distribution</td>
<td>50 us</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
<td>HPUX 9, 10, 11</td>
<td><a href="http://us-support.external.hp.com">http://us-support.external.hp.com</a></td>
<td>50 us</td>
</tr>
<tr>
<td>Various</td>
<td>Free BSD</td>
<td>NTP Distribution</td>
<td>20 us</td>
</tr>
</tbody>
</table>

Attention: Thunderbolt Receiver has not being tested on the previous software and hardware plataforms.

10.20.4 GPS Receiver

The Palisade GPS receiver is an 8-channel smart antenna, housing the GPS receiver, antenna and interface in a single unit, and is designed for rooftop deployment in static timing applications.

Palisade generates a PPS synchronized to UTC within +/- 100 ns. The Palisade’s external event input with 40 nanosecond resolution is utilized by the Palisade NTP driver for asynchronous precision time transfer.

No user initialization of the receiver is required. This driver is compatible with the following versions of Palisade:

<table>
<thead>
<tr>
<th>Version</th>
<th>Event Input</th>
<th>Trimble Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.02</td>
<td>No</td>
<td>26664-00</td>
</tr>
<tr>
<td>7.02E</td>
<td>Yes</td>
<td>26664-10</td>
</tr>
<tr>
<td>7.12</td>
<td>Yes</td>
<td>38158-00</td>
</tr>
</tbody>
</table>

Note: When using Palisade 26664-00, you must set fudge flag2 to 1 in ntp.conf. See configuration.

GPS Installation

A location with unobstructed view of the horizon is recommended. Palisade is designed to be securely mounted atop standard 3/4 inch threaded pipe.

The 12 conductor (dia. 10 mm) power and I/O cable must be routed from the rooftop site to the NTP server and properly strain relieved.
GPS Connection

The Palisade is equipped with dual (A & B) RS-422 serial interfaces and a differential TTL PPS output. An RS-232 / RS-422 Interface Module is supplied with the Palisade NTP Synchronization Kit. Palisade port A must be connected to the NTP host server. Maximum antenna cable length is 500 meters. See the pinouts table for detailed connection information.

Palisade’s port B provides a TSIP (Trimble Standard Interface Protocol) interface for diagnostics, configuration, and monitoring. Port B and the PPS output are not currently used by the Palisade NTP reference clock driver.

10.20.5 O/S Serial Port Configuration

The driver attempts to open the device /dev/palisade<u> where <u> is the NTP refclock unit number as defined by the LSB of the refclock address. Valid refclock unit numbers are 0 - 3.

The user is expected to provide a symbolic link to an available serial port device. This is typically performed by a command such as:

```
ln -s /dev/ttyS0 /dev/palisade0
```

Windows NT does not support symbolic links to device files. COMx is used by the driver, based on the refclock unit number, where unit 1 corresponds to COM1: and unit 3 corresponds to COM3:

10.20.6 NTP Configuration

Palisade NTP configuration file ntp.conf with event polling:

```
# The Primary reference
server 127.127.29.0 # Trimble Palisade GPS Refclock Unit #0
peer terrapin.csc.ncsu.edu # internet server
# Drift file for expedient re-synchronization after downtime or reboot.
driftfile /etc/ntp.drift
```

Configuration without event polling:

```
# The Primary reference
server 127.127.29.0 # Trimble Palisade GPS (Stratum 1).
# Set packet delay
fudge 127.127.29.0 time1 0.020
# and set flag2 to turn off event polling.
fudge 127.127.29.0 flag2 1
```

Thunderbolt NTP Configuration file

Configuration without event polling:

```
# The Primary reference
server 127.127.29.0 mode 2 # Trimble Thunderbolt GPS (Stratum 1).
# Set packet delay
fudge 127.127.29.0 time1 0.020
# and set flag2 to turn off event polling.
```
Currently the Thunderbolt mode doesn’t support event polling, the reasons are explained on the “Notes on the Thunderbolt Receiver’s Firmware” section at the end of this documentation.

### 10.20.7 Time Transfer and Polling

Time transfer to the NTP host is performed via the Palisade’s comprehensive time packet output. The time packets are output once per second, and whenever an event timestamp is requested.

The driver requests an event time stamp at the end of each polling interval, by pulsing the RTS (request to send) line on the serial port. The Palisade GPS responds with a time stamped event packet.

Time stamps are reported by the Palisade with respect to UTC time. The GPS receiver must download UTC offset information from GPS satellites. After an initial UTC download, the receiver will always start with correct UTC offset information.

### 10.20.8 Run NTP in Debugging Mode

The following procedure is recommended for installing and testing a Palisade NTP driver:

1. Perform initial checkout procedures. Place the GPS receiver outdoors; with clear view of the sky. Allow the receiver to obtain an UTC almanac.
2. Verify presence of timing packets by observing the 1 Hz (PPS) led on the interface module. It should flash once per second.
3. Connect Palisade’s port A to the NTP host.
4. Configure NTP and the serial I/O port on the host system.
5. Initially use fudge flag2 in ntp.conf, to disable event polling (see configuration).
6. Run NTP in debug mode (-d -d), to observe Palisade_receieve events.
7. The driver reports the tracking status of the receiver. Make sure it is tracking several satellites.
8. Remove fudge flag2 and restart ntpd in debug mode to observe palisade_receive events.
9. If event polling fails, verify the connections and that the host hardware supports RTS control.

### 10.20.9 Event Logging

System and Event log entries are generated by NTP to report significant system events. Administrators should monitor the system log to observe NTP error messages. Log entries generated by the Palisade NTP reference clock driver will be of the form:

```
Nov 14 16:16:21 terrapin ntpd[1127]: Palisade #0: message
```

### 10.20.10 Fudge Factors

- **time1 <time>** Specifies the time offset calibration factor, in seconds and fraction, with default 0.0. If event capture is not used, time1 should be set to 20 milliseconds to correct serial line and operating system delays incurred in capturing time stamps from the synchronous packets.

- **stratum <number>** Specifies the driver stratum, in decimal from 0 to 15, with default 0.
**refid <string>** Specifies the driver reference identifier, GPS.

**flag2 0 | 1** When set to 1, driver does not use hardware event capture. The synchronous packet output by the receiver at the beginning of each second is time stamped by the driver. If triggering the event pulse fails, the driver falls back to this mode automatically.

### 10.20.11 Mode Parameter

**mode number** The mode parameter to the server command specifies the specific hardware this driver is for. The default is 0 for a normal Trimble Palisade. The other options are 1 for an Endrun Praecis in Trimble emulation mode, and 2 for the Trimble Thunderbolt GPS Disciplined Clock Receiver.

### 10.20.12 DEFINEs

The following constants are defined in the driver source code. These defines may be modified to improve performance or adapt to new operating systems.

<table>
<thead>
<tr>
<th>Label</th>
<th>Definition</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE</td>
<td>The serial port device to be used by the driver</td>
<td>/dev/palisade</td>
</tr>
<tr>
<td>PRECISION</td>
<td>Accuracy of time transfer</td>
<td>1 microsecond</td>
</tr>
<tr>
<td>CURRENT.UTC</td>
<td>Valid GPS - UTC offset</td>
<td>13</td>
</tr>
<tr>
<td>SPEED232</td>
<td>Host RS-232 baud rate</td>
<td>B9600</td>
</tr>
<tr>
<td>TRMB_MINPOLL</td>
<td>Minimum polling interval</td>
<td>5 (32 seconds)</td>
</tr>
<tr>
<td>TRMB_MAXPOLL</td>
<td>Maximum interval between polls</td>
<td>7 (128 seconds)</td>
</tr>
</tbody>
</table>

### 10.20.13 Data Format

Palisade port A can output two synchronous time packets. The NTP driver can use either packet for synchronization. Packets are formatted as follows:

**Packet 8F-AD (Primary NTP Packet)**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Item</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sub-Packet ID</td>
<td>BYTE</td>
<td>Subcode 0xAD</td>
</tr>
<tr>
<td>1 - 2</td>
<td>Event Count</td>
<td>INTEGER</td>
<td>External event count recorded (0 = PPS)</td>
</tr>
<tr>
<td>3 - 10</td>
<td>Fractional Second</td>
<td>DOUBLE</td>
<td>Time elapsed in current second (s)</td>
</tr>
<tr>
<td>11</td>
<td>Hour</td>
<td>BYTE</td>
<td>Hour (0 - 23)</td>
</tr>
<tr>
<td>12</td>
<td>Minute</td>
<td>BYTE</td>
<td>Minute (0 - 59)</td>
</tr>
<tr>
<td>13</td>
<td>Second</td>
<td>BYTE</td>
<td>Second (0 - 59; 60 = leap)</td>
</tr>
<tr>
<td>14</td>
<td>Day</td>
<td>BYTE</td>
<td>Date (1 - 31)</td>
</tr>
<tr>
<td>15</td>
<td>Month</td>
<td>BYTE</td>
<td>Month (1 - 12)</td>
</tr>
<tr>
<td>16 - 17</td>
<td>Year</td>
<td>INTEGER</td>
<td>Year (4 digit)</td>
</tr>
<tr>
<td>18</td>
<td>Receiver Status</td>
<td>BYTE</td>
<td>Tracking Status</td>
</tr>
<tr>
<td>19</td>
<td>UTC Flags</td>
<td>BYTE</td>
<td>Leap Second Flags</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
<td>BYTE</td>
<td>Contains 0xFF</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td>BYTE</td>
<td>Contains 0xFF</td>
</tr>
</tbody>
</table>

**Leap Second Flag Definition:**

Bit 0: (1) UTC Time is available
Bits 1 - 3: Undefined
Bit 4: (1) Leap Scheduled: Leap second pending asserted by GPS control segment.
Bit 5: (1) Leap Pending: set 24 hours before, until beginning of leap second.
Bit 6: (1) GPS Leap Warning: 6 hours before until 6 hours after leap event
Bit 7: (1) Leap In Progress. Only set during the leap second.

Tracking Status Flag Definitions:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Accuracy</th>
<th>Receiver Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Receiver is Navigating</td>
<td>+/- 1 us</td>
<td>Self Survey</td>
</tr>
<tr>
<td>1</td>
<td>Static 1 Sat. Timing Mode</td>
<td>+/- 1 us</td>
<td>1-D Timing</td>
</tr>
<tr>
<td>2</td>
<td>Approximate Time</td>
<td>20 - 50 ms</td>
<td>Acquisition</td>
</tr>
<tr>
<td>3</td>
<td>Startup</td>
<td>N/A</td>
<td>Initialization</td>
</tr>
<tr>
<td>4</td>
<td>Startup</td>
<td>N/A</td>
<td>Initialization</td>
</tr>
<tr>
<td>5</td>
<td>Dilution of Position too High</td>
<td>5 ppm</td>
<td>Self Survey</td>
</tr>
<tr>
<td>6</td>
<td>Static 1 Sat. Timing: Sat. not usable</td>
<td>5 ppm</td>
<td>1-D Timing</td>
</tr>
<tr>
<td>7</td>
<td>No Satellites Usable</td>
<td>N/A</td>
<td>Self Survey</td>
</tr>
<tr>
<td>8</td>
<td>Only 1 Satellite Usable</td>
<td>20 - 50 ms</td>
<td>Self Survey</td>
</tr>
<tr>
<td>9</td>
<td>Only 2 Satellite Usable</td>
<td>20 - 50 ms</td>
<td>Self Survey</td>
</tr>
<tr>
<td>10</td>
<td>Only 3 Satellites Usable</td>
<td>20 - 50 ms</td>
<td>Self Survey</td>
</tr>
<tr>
<td>11</td>
<td>Invalid Solution</td>
<td>N/A</td>
<td>Error</td>
</tr>
<tr>
<td>12</td>
<td>Differential Corrections</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>13</td>
<td>Overdetermined Fixes</td>
<td>+/- 100 ns</td>
<td>Timing Steady State</td>
</tr>
</tbody>
</table>
### Packet 8F-0B (Comprehensive Timing Packet)

<table>
<thead>
<tr>
<th>Byte</th>
<th>Item</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sub-Packet ID</td>
<td>BYTE</td>
<td>Subcode 0x0B</td>
</tr>
<tr>
<td>1 - 2</td>
<td>Event Count</td>
<td>INT</td>
<td>External event count recorded (0 = PPS)</td>
</tr>
<tr>
<td>3 - 10</td>
<td>UTC / GPS TOW</td>
<td>DOUBLE</td>
<td>UTC / GPS time of week (seconds)</td>
</tr>
<tr>
<td>11</td>
<td>Date</td>
<td>BYTE</td>
<td>Day of Month</td>
</tr>
<tr>
<td>12</td>
<td>Month</td>
<td>BYTE</td>
<td>Month of Event</td>
</tr>
<tr>
<td>13 - 14</td>
<td>Year</td>
<td>INT</td>
<td>Year of event</td>
</tr>
<tr>
<td>15</td>
<td>Receiver Mode</td>
<td>BYTE</td>
<td>Receiver operating dimensions: 0: Horizontal (2D) 1: Full Position (3D) 2: Single Satellite (0D) 3: Automatic (2D / 3D) 4: DGPS reference 5: Clock hold (2D) 6: Over determined Clock</td>
</tr>
<tr>
<td>15 - 17</td>
<td>UTC Offset</td>
<td>INT</td>
<td>UTC Offset value (seconds)</td>
</tr>
<tr>
<td>18 - 25</td>
<td>Oscillator Bias</td>
<td>DOUBLE</td>
<td>Oscillator BIAS (meters)</td>
</tr>
<tr>
<td>26 - 33</td>
<td>Oscillator Drift Rate</td>
<td>DOUBLE</td>
<td>Oscillator Drift (meters / second)</td>
</tr>
<tr>
<td>34 - 37</td>
<td>Bias Uncertainty</td>
<td>SINGLE</td>
<td>Oscillator bias uncertainty (meters)</td>
</tr>
<tr>
<td>38 - 41</td>
<td>Drift Uncertainty</td>
<td>SINGLE</td>
<td>Oscillator bias rate uncertainty (m / sec)</td>
</tr>
<tr>
<td>42 - 49</td>
<td>Latitude</td>
<td>DOUBLE</td>
<td>Latitude in radians</td>
</tr>
<tr>
<td>50 - 57</td>
<td>Longitude</td>
<td>DOUBLE</td>
<td>Longitude in radians</td>
</tr>
<tr>
<td>58 - 65</td>
<td>Altitude</td>
<td>DOUBLE</td>
<td>Altitude above mean sea level, in meters</td>
</tr>
<tr>
<td>66 - 73</td>
<td>Satellite ID</td>
<td>BYTE</td>
<td>SV Id No. of tracked satellites</td>
</tr>
</tbody>
</table>

**Thunderbolt Timing packets Data Format**

Thunderbolt can output 2 synchronous packets.
Primary Timing Packet - 0x8FAB

<table>
<thead>
<tr>
<th>Byte</th>
<th>Bit</th>
<th>Item</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>Sub-code</td>
<td>UINT8</td>
<td>0xAB</td>
<td></td>
</tr>
<tr>
<td>1-4</td>
<td></td>
<td>Time of Week</td>
<td>UINT32</td>
<td></td>
<td>GPS seconds of week</td>
</tr>
<tr>
<td>5-6</td>
<td></td>
<td>Week Number</td>
<td>UINT16</td>
<td></td>
<td>GPS Week Number</td>
</tr>
<tr>
<td>7-8</td>
<td></td>
<td>UTC Offset</td>
<td>SINT16</td>
<td></td>
<td>UTC Offset (seconds)</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Timing Flag</td>
<td>Bit field</td>
<td>0 or 1</td>
<td>GPS Time or UTC Time</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>0 or 1</td>
<td>GPS PPS or UTC PPS</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>0 or 1</td>
<td>time is set or time is not set</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>0 or 1</td>
<td>have UTC info or no UTC info</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>0 or 1</td>
<td>Time from GPS or time from user</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Seconds</td>
<td>UINT8</td>
<td>0-59</td>
<td>(60 for UTC leap second event)</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Minutes</td>
<td>UINT8</td>
<td>0-59</td>
<td>Minutes of Hour</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>Hours</td>
<td>UINT8</td>
<td>0-23</td>
<td>Hour of Day</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Day of Month</td>
<td>UINT8</td>
<td>1-31</td>
<td>Day of Month</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>Month</td>
<td>UINT8</td>
<td>1-12</td>
<td>Month of Year</td>
</tr>
<tr>
<td>15-16</td>
<td></td>
<td>Year</td>
<td>UINT16</td>
<td></td>
<td>Four digits of Year (e.g. 1998)</td>
</tr>
</tbody>
</table>
## Supplemental Timing Packet - 0x8FAC

<table>
<thead>
<tr>
<th>Byte</th>
<th>Bit</th>
<th>Item</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>Sub-code</td>
<td>UINT8</td>
<td>0xAC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Receiver Mode</td>
<td>UINT8</td>
<td></td>
<td>Automatic (2D/3D)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>Single Satellite (Time)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td>Horizontal (2D)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>Full Position (3D)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td>DGPS Reference</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td>Clock Hold (2D)</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
<td>Overdetermined Clock</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Disciplining Mode</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td>Power-Up</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>Auto Holdover</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td>Manual Holdover</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td>Recovery</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
<td>Not Used</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Self-Survey Progress</td>
<td>UINT8</td>
<td>0-100%</td>
<td>Disciplining disabled</td>
</tr>
<tr>
<td>4-7</td>
<td></td>
<td>Holdover Duration</td>
<td>UINT32</td>
<td></td>
<td>seconds</td>
</tr>
<tr>
<td>8-9</td>
<td></td>
<td>Critical Alarms</td>
<td>UINT16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>ROM checksum error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>RAM check has failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>Power supply failure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>FPGA check has failed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>Oscillator control voltage at rail</td>
<td></td>
</tr>
<tr>
<td>10-11</td>
<td></td>
<td>Minor Alarms</td>
<td>UINT16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>Power-Up</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>Auto Holdover</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>Manual Holdover</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>Recovery</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>Disciplining disabled</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>GPS Decoding Status</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>Doing fixes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>Don t have GPS time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>PDOP is too high</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>No usable sats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>Only 1 usable sat</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>Only 2 usable sats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td>Only 3 usable sats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td>The chosen sat is unusable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td>TRAIM rejected the fix</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>Disciplining Activity</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>Phase locking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>Oscillator warming up</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>Frequency locking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>Placing PPS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>Initializing loop filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>Compensating OCXO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>Inactive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td>Recovery mode</td>
<td></td>
</tr>
<tr>
<td>226</td>
<td></td>
<td>Spare Status 1</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>Spare Status 2</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Spare Status 3</td>
<td>UINT8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10.20.14 Pinouts

The following connections are required when connecting Palisade with a host:

<table>
<thead>
<tr>
<th>Description</th>
<th>Host</th>
<th>Palisade</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Port A</strong></td>
<td>DB-9</td>
<td>DB-25</td>
</tr>
<tr>
<td>Receive Data</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Request to Send</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Signal Ground</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td><strong>Port B</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Data</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Transmit Data</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Signal Ground</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Note: If driving the RS-422 inputs on the Palisade single ended, i.e. using the Green and Gray connections only, does not work on all serial ports. Use of the Palisade NTP Synchronization Interface Module is recommended.

The 12 pin connector pinout definition: Face the round 12 pin connector at the end of the cable, with the notch turned upwards. Pin 1 is to the left of the notch. Pins 2 - 8 wrap around the bottom, counterclockwise to pin 9 on the right of the notch. Pin 10 is just below the notch. Pins 10 (top), 11 (bottom left) and 12 (bottom right) form a triangle in the center of the connector.

Pinouts for the Palisade NTP host adapter (Trimble PN 37070) DB-25 M connector are as follows:

<table>
<thead>
<tr>
<th>DB-25M</th>
<th>Conductor</th>
<th>Palisade</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>1</td>
<td>Power</td>
</tr>
<tr>
<td>7</td>
<td>Black</td>
<td>9</td>
<td>Ground</td>
</tr>
<tr>
<td>9</td>
<td>Black/White</td>
<td>12</td>
<td>PPS -</td>
</tr>
<tr>
<td>10</td>
<td>Green</td>
<td>8</td>
<td>Transmit Port A (T-)</td>
</tr>
<tr>
<td>11</td>
<td>Brown</td>
<td>4</td>
<td>Transmit Port B (T-)</td>
</tr>
<tr>
<td>12</td>
<td>Gray</td>
<td>7</td>
<td>Receive Port A (R+)</td>
</tr>
<tr>
<td>13</td>
<td>Orange</td>
<td>3</td>
<td>Receive Port B (R+)</td>
</tr>
<tr>
<td>21</td>
<td>Orange/White</td>
<td>11</td>
<td>PPS +</td>
</tr>
<tr>
<td>22</td>
<td>Blue</td>
<td>10</td>
<td>Transmit Port A (T+)</td>
</tr>
<tr>
<td>23</td>
<td>Yellow</td>
<td>5</td>
<td>Transmit Port B (T+)</td>
</tr>
<tr>
<td>24</td>
<td>White</td>
<td>6</td>
<td>Receive Port A (R-)</td>
</tr>
<tr>
<td>25</td>
<td>Violet</td>
<td>2</td>
<td>Receive Port B (R-)</td>
</tr>
</tbody>
</table>

Notes on the Thunderbolt Receiver’s Firmware

The support for Thunderbolt Receiver in the palisade driver doesn’t support (for now) event-polling, the reason is that the Thunderbolt receiver the patch is written for doesn’t support time-on-request, so you just have to sit there and wait for the time to arrive with the PPS. We tried to contact Trimble because there’s presumably a firmware update that support it, but we didn’t have much luck. Here is a link explaining the situation:


Questions or Comments:

Sven Dietrich
Trimble Navigation Ltd.
10.21 Motorola Oncore GPS receiver

10.21.1 Synopsis

Address: 127.127.30.u
Reference ID: GPS
Driver ID: ONCORE
Serial Port: /dev/oncore.serial.u; 9600 baud, 8-bits, no parity.
PPS Port: /dev/oncore.pps.u; PPS_CAPTUREASSERT required, PPS_OFFSETASSERT supported.
Configuration File: ntp.oncore, or ntp.oncore.u, or ntp.oncoreu, in /etc/ntp or /etc.

10.21.2 Description

This driver supports most models of the Motorola Oncore GPS receivers (Basic, PVT6, VP, UT, UT+, GT, GT+, SL, M12, M12+T), as long as they support the Motorola Binary Protocol.

The interesting versions of the Oncore are the VP, the UT+, the “Remote” which is a prepackaged UT+, and the M12 Timing. The VP is no longer available new, and the UT, GT, and SL are at end-of-life. The Motorola evaluation kit can be recommended. It interfaces to a PC straightaway, using the serial (DCD) or parallel port for PPS input and packs the receiver in a nice and sturdy box. Less expensive interface kits are available from TAPR and Synergy.

The driver requires a standard PPS interface for the pulse-per-second output from the receiver. The serial data stream alone does not provide precision time stamps (0-50msec variance, according to the manual), whereas the PPS output is precise down to 50 nsec (1 sigma) for the VP/UT models and 25 nsec for the M12 Timing. If you do not have the PPS signal available, then you should probably be using the NMEA driver rather than the Oncore driver. Most of these are available on-line.

The driver will use the “position hold” mode with user provided coordinates, the receivers built-in site-survey, or a similar algorithm implemented in this driver to determine the antenna position.
10.21.3 Monitor Data

The driver always puts a lot of useful information on the clockstats file, and when run with debugging can be quite chatty on stdout. When first starting to use the driver you should definitely review the information written to the clockstats file to verify that the driver is running correctly.

In addition, on platforms supporting Shared Memory, all of the messages received from the Oncore receiver are made available in shared memory for use by other programs. See the Oncore-SHMEM manual page for information on how to use this option. For either debugging or using the SHMEM option, an Oncore Reference Manual for the specific receiver in use will be required.

10.21.4 Fudge Factors

\texttt{time1 time} Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

\texttt{time2 time} Not used by this driver.

\texttt{stratum number} Specifies the driver stratum, in decimal from 0 to 15, with default 0.

\texttt{refid string} Specifies the driver reference identifier, an ASCII string from one to four characters, with default \texttt{GPS}.

\texttt{flag1 0 | 1} Not used by this driver.

\texttt{flag2 0 | 1} Not used by this driver.

\texttt{flag3 0 | 1} Not used by this driver.

\texttt{flag4 0 | 1} Not used by this driver.

10.21.5 Additional Information

The driver was initially developed on FreeBSD, and has since been tested on Linux, SunOS and Solaris.

Configuration

There is a driver specific configuration file \texttt{ntp.oncore} (or \texttt{ntp.oncore.u} or \texttt{ntp.oncoreu} if you must distinguish between more than one Oncore receiver) that contains information on the startup mode, the location of the GPS receiver, an offset of the PPS signal from zero, and the cable delay. The offset shifts the PPS signal to avoid interrupt pileups ‘on’ the second, and adjusts the timestamp accordingly. See the driver source for information on this file. The default with no file is: no delay, no offset, and a site survey is done to get the location of the gps receiver.

The following three options can be set in the driver specific configuration file only if the driver is using the PPSAPI. The edge of the PPS signal that is ‘on-time’ can be set with the keywords [ASSERT/CLEAR] and the word HARDPPS will cause the PPS signal to control the kernel PLL.

Performance

Really good. With the VP/UT+, the generated PPS pulse is referenced to UTC(GPS) with better than 50 nsec (1 sigma) accuracy. The limiting factor will be the timebase of the computer and the precision with which you can timestamp the rising flank of the PPS signal. Using FreeBSD, a FPGA based Timecounter/PPS interface, and an ovenized quartz oscillator, that performance has been reproduced. For more details on this aspect: Sub-Microsecond timekeeping under FreeBSD.
10.22 Rockwell Jupiter GPS receiver

10.22.1 Synopsis

Address: 127.127.31.u
Reference ID: GPS
Driver ID: JUPITER
Serial Port: /dev/gpsu; 9600 baud, 8-bits, no parity.

10.22.2 Description

This driver supports at least some models of the Rockwell Jupiter GPS receivers (Jupiter 11, Jupiter-T), they must at least support the Zodiac Binary Protocol.

The driver requires a standard PPS interface for the pulse-per-second output from the receiver. The serial data stream alone does not provide precision time stamps, whereas the PPS output is precise down to 40 ns (1 sigma) for the Jupiter 11 and 25 ns (1 sigma) for Jupiter-T according to the documentation. If you do not have the PPS signal available, then you should probably not be using the Jupiter receiver as a time source. This driver requires a PPS signal and the time output from Jupiter receivers is not predictable in NMEA mode; the reported time can take one second steps.

10.22.3 Monitor Data

The driver always puts a lot of useful information on the clockstats file, and when run with debugging can be quite chatty on stdout. When first starting to use the driver you should definitely review the information written to the clockstats file to verify that the driver is running correctly.

10.22.4 Fudge Factors

\texttt{time1 time} Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
\texttt{time2 time} Not used by this driver. Should be left at zero.
\texttt{stratum number} Specifies the driver stratum, in decimal from 0 to 15, with default 0.
\texttt{refid string} Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPS.
\texttt{flag1 0 | 1} Not used by this driver.
\texttt{flag2 0 | 1} Specifies the mobility of the GPS receiver: 0 for walking (default), 1 for driving.
\texttt{flag3 0 | 1} Specifies the PPS signal on-time edge: 0 for assert (default), 1 for clear.
\texttt{flag4 0 | 1} Not used by this driver.

10.22.5 Additional Information

The driver was resurrected from a sorry state using the Windows NT port and a Jupiter 11, and has since seen little testing on other platforms. On Windows there exist a barrier though, as there is no publicly available PPSAPI implementation, at least not to my knowledge. However, there has been one success report using Linux 2.4.20 and PPSkit 2.1.1.
The Jupiter receivers seem to have quite a few names. They are referred to at least as Rockwell receivers, Navman receivers, Zodiac receivers, Conexant receivers and SiRF Technology receivers. Rockwell seems to be the original and most commonly used name and Navman seems to be the current supplier.

**Configuration**

The edge of the PPS signal that is ‘on-time’ can be set with flag2. There is currently no way to cause the PPS signal to control the kernel PLL.

**Performance**

The performance is largely unexplored. I have achieved submillisecond stability using a Jupiter 11, but the poor result is more than likely due to the proprietary PPSAPI implementation or Windows itself.

This driver does not handle leap seconds.

### 10.23 Chrono-log K-series WWVB receiver

#### 10.23.1 Synopsis

Address: 127.127.32.u

Reference ID: CHRONOLOG

Driver ID: CHRONOLOG

Serial Port: `/dev/chronologu`; 2400 bps, 8-bits, no parity

Features: (none)

#### 10.23.2 Description

This driver supports the Chrono-log K-series WWVB receiver. This is a very old receiver without provisions for leap seconds, quality codes, etc. It assumes output in the local time zone, and that the C library mktime()/localtime() routines will correctly convert back and forth between local and UTC. There is a hack in the driver for permitting UTC, but it has not been tested.

Most of this code is originally from refclock_wwvb.c with thanks. It has been so mangled that wwvb is not a recognizable ancestor.

**Timecode format:** Y yy/mm/ddCLZhh:mm:ssCL

- Y - year/month/date line indicator
- yy/mm/dd -- two-digit year/month/day
- C - \r (carriage return)
- L - \n (newline)
- Z - timestamp indicator
- hh:mm:ss - local time

#### 10.24 Dumb Clock

#### 10.24.1 Synopsis

Address: 127.127.33.u

Reference ID: DUMBCLOCK

Driver ID: DUMBCLOCK
10.24.2 Description

This driver supports a dumb ASCII clock that only emits localtime at a reliable interval. This has no provisions for leap seconds, quality codes, etc. It assumes output in the local time zone, and that the C library mktime()/localtime() routines will correctly convert back and forth between local and UTC.

Most of this code is originally from refclock_wwvb.c with thanks. It has been so mangled that wwvb is not a recognizable ancestor.

Timecode format: hh:mm:ssCL
hh:mm:ss - local time
C - \r (carriage return)
L - \n (newline)

10.25 Ultralink Clock

10.25.1 Synopsis

Address: 127.127.34.u
Reference ID: WWVB
Driver ID: ULINK
Serial Port: /dev/wwvbu; 9600 bps, 8-bits, no parity
Features: (none)

10.25.2 Description

This driver supports the Ultralink Model 325 (replacement for Model 320) RS-232 powered WWVB receiver. PDF specs available on http://www.ulio.com/. This driver also supports the Model 320, 330,331,332 decoders in both polled or continuous time code mode. Leap second and quality are supported. Most of this code is originally from refclock_wwvb.c with thanks. Any mistakes are mine. Any improvements are welcome.

10.25.3 Model 325 timecode format

<cr><lf>RQ_1C00LYYYY+DDDUTCS_HH:MM:SSL+5

R = Signal readability indicator, ranging from R1 to R5 Q R1 is unreadable, R5 is best reception
_ = Space
I = prev. received data bit, values: 0, 1 ,M or ? unknown C = Signal reception from (C)olorado or (H)awaii 0 = Hours since last WWVB time and flag code update, values 0 00 to 99 (hopefully always 00)
L = HEX A5 if receiver is locked to WWVB, Space if not
YYYY = Year from 2000 to 2099
+ = ‘+’ if current year is a leap year, else ‘ ‘
DDD = current day in the year from 1 to 365/366
UTC = timezone (always UTC)
S = Daylight savings indicator, (S)TD, (D)ST, (O) transition into DST, (I) transition out of DST
_ = Space
HH = UTC hour 0 to 23
: = Time delimiter, ‘.’ if synced, Space if not
MM = Minutes of current hour from 0 to 59
: = Time delimiter, ‘.’ if synced, Space if not
SS = Seconds of current minute from 0 to 59
mm = 10's milliseconds of the current second from 00 to 99
L = Leap second pending at end of month, (I)nsert, (D)elete or Space
+5 = UT1 correction, +/- .1 sec increments

Note that Model 325 reports a very similar output like Model 33X series. The driver for this clock is similar to Model 33X behavior. On a unmodified new ULM325 clock, the polling flag (flag1 =1) needs to be set.

### 10.25.4 Model 320 timecode format

```
<cr><lf>SQRYYYYDDD+HH:MM:SS.mmLT<cr>
```

S = ‘S’ – sync’d in last hour, ‘0’-'9' - hours x 10 since last update, else ‘?’
Q = Number of correlating time-frames, from 0 to 5
R = ‘R’ – reception in progress,’N’ – Noisy reception, ‘ ’ – standby mode
YYYY = year from 1990 to 2089
DDD = current day from 1 to 366 + = ‘+’ if current year is a leap year, else ‘ ’
HH = UTC hour 0 to 23
MM = Minutes of current hour from 0 to 59
SS = Seconds of current minute from 0 to 59
mm = 10's milliseconds of the current second from 00 to 99
L = Leap second pending at end of month – ‘I’ = insert, ‘D’=delete
T = DST <-> STD transition indicators

Note that this driver does not do anything with the T flag. The M320 also has a ‘U’ command which returns UT1 correction information. It is not used in this driver.

### 10.25.5 Model 33x timecode format

```
S9+D 00 YYYY+DDDTUSC HH:MM:SS+5
```

S = sync indicator S insync N not in sync the sync flag is WWVB decoder sync nothing to do with time being correct

9+ = signal level 0 thru 9+ If over 9 indicated as 9
D = data bit (fun to watch but useless ;-) space
00 = hours since last GOOD WWVB frame sync space
YYYY = current year + = leap year indicator
DDD = day of year
UTC = timezone (always UTC)
S = daylight savings indicator space
HH = hours : = This is the REAL in sync indicator (: = insync)
MM = minutes : = = in sync ? = NOT in sync
SS = seconds
L = leap second flag
+5 = UT1 correction (sign + digit ))

This driver ignores UT1 correction, DST indicator, Leap year and signal level.

10.25.6 Fudge factors

flag1 polling enable (1=poll 0=no poll)

10.26 Conrad parallel port radio clock

10.26.1 Synopsis

Address: 127.127.35.\text{u}
Reference ID: PCF
Driver ID: PCF
Parallel Port: /dev/pcfclocks/u or /dev/pcfclocku

10.26.2 Description

This driver supports the parallel port radio clock sold by Conrad Electronic under order numbers 967602 and 642002. This clock is put between a parallel port and your printer. It receives the legal German time, which is either CET or CEST, from the DCF77 transmitter and uses it to set its internal quartz clock. The DCF77 transmitter is located near Frankfurt/Main and covers a radius of more than 1500 kilometers.

The pcfclock device driver is required in order to use this reference clock driver. Currently device drivers for \text{Linux} and FreeBSD are available.

This driver uses C library functions to convert the received timecode to UTC and thus requires that the local timezone be CET or CEST. If your server is not located in Central Europe you have to set the environment variable TZ to CET before starting ndpd.

10.26.3 Monitor Data

Each timecode is written to the clockstats file in the format YYYY MM DD HH MI SS.

10.26.4 Fudge Factors

\text{time1} time Specifies the time offset calibration factor, in seconds and fraction, with default 0.1725.
\text{time2} time Not used by this driver.
\text{stratum} number Specifies the driver stratum, in decimal from 0 to 15, with default 0.
**refid string**

Specifies the driver reference identifier, an ASCII string from one to four characters, with default PCF.

**flag1** 0 | 1 Not used by this driver.

**flag2** 0 | 1 If set to 1, the radio clock’s synchronisation status bit is ignored, i.e., the timecode is used without a check.

**flag3** 0 | 1 Not used by this driver.

**flag4** 0 | 1 Not used by this driver.

### 10.27 Radio WWV/H Audio Demodulator/Decoder

#### 10.27.1 Synopsis

Address: 127.127.36.u
Reference ID: WVf or WHf
Driver ID: WWV_AUDIO
Autotune Port: `/dev/icom`; 1200/9600 baud, 8-bits, no parity
Audio Device: `/dev/audio` and `/dev/audioctl`

#### 10.27.2 Description

This driver synchronizes the computer time using shortwave radio transmissions from NIST time/frequency stations WWV in Ft. Collins, CO, and WWVH in Kauai, HI. Transmissions are made continuously on 2.5, 5, 10 and 15 MHz from both stations and on 20 MHz from WWV. An ordinary shortwave receiver can be tuned manually to one of these frequencies or, in the case of ICOM receivers, the receiver can be tuned automatically by the driver as propagation conditions change throughout the day and season. The radio is connected via an optional attenuator and cable to either the microphone or line-in port of a workstation or PC.

The driver requires an audio codec or sound card with sampling rate 8 kHz and μ-law companding to demodulate the data. This is the same standard as used by the telephone industry and is supported by most hardware and operating systems, including Solaris, FreeBSD and Linux, among others. In this implementation only one audio driver and codec can be supported on a single machine. In order to assure reliable signal capture, the codec frequency error must be less than 187 PPM (.0187 percent). If necessary, the `tinker codec` configuration command can be used to bracket the codec frequency to this range.

In general and without calibration, the driver is accurate within 1 ms relative to the broadcast time when tracking a station. However, variations up to 0.3 ms can be expected due to diurnal variations in ionospheric layer height and ray geometry. In Newark DE, 2479 km from the transmitter, the predicted two-hop propagation delay varies from 9.3 ms in sunlight to 9.0 ms in moonlight. When not tracking the station the accuracy depends on the computer clock oscillator stability, ordinarily better than 0.5 PPM.

After calibration relative to the PPS signal from a GPS receiver, the mean offset with a 2.4-GHz P4 running FreeBSD 6.1 is generally within 0.1 ms short-term with 0.4 ms jitter. The long-term mean offset varies up to 0.3 ms due to propagation path geometry variations. The processor load due to the driver is 0.4 percent on the P4.

The driver performs a number of error checks to protect against overdriven or underdriven input signal levels, incorrect signal format or improper hardware configuration. The specific checks are detailed later in this page. Note that additional checks are done elsewhere in the reference clock interface routines.

This driver incorporates several features in common with other audio drivers such as described in the Radio CHU Audio Demodulator/Decoder and the IRIG Audio Decoder pages. They include automatic gain control (AGC), selectable...
audio codec port and signal monitoring capabilities. For a discussion of these common features, as well as a guide to hook up, debugging and monitoring, see the Reference Clock Audio Drivers page.

10.27.3 Technical Overview

The driver processes 8-kHz $\mu$-law companded codec samples using maximum-likelihood techniques which exploit the considerable degree of redundancy available in the broadcast signal. The WWV signal format is described in NIST Special Publication 432 (Revised 1990) and also available on the WWV/H web site. It consists of three elements, a 5-ms, 1000-Hz pulse, which occurs at the beginning of each second, a 800-ms, 1000-Hz pulse, which occurs at the beginning of each minute, and a pulse-width modulated 100-Hz subcarrier for the data bits, one bit per second. The WWVH format is identical, except that the 1000-Hz pulses are sent at 1200 Hz. Each minute encodes nine BCD digits for the time of century plus seven bits for the daylight savings time (DST) indicator, leap warning indicator and DUT1 correction.

The demodulation and decoding algorithms used by this driver are based on a machine language program developed for the TAPR DSP93 DSP unit, which uses the TI 320C25 DSP chip. The analysis, design and performance of the program for this unit is described in: Mills, D.L. A precision radio clock for WWV transmissions. Electrical Engineering Report 97-8-1, University of Delaware, August 1997, 25 pp. Available from http://www.eecis.udel.edu/~mills/papers.html. For use in this driver, the original program was rebuilt in the C language and adapted to the NTP driver interface. The algorithms have been modified to improve performance, especially under weak signal conditions and to provide an automatic frequency and station selection feature.

As in the original program, the clock discipline is modelled as a Markov process, with probabilistic state transitions corresponding to a conventional clock and the probabilities of received decimal digits. The result is a performance level with very high accuracy and reliability, even under conditions when the minute beep of the signal, normally its most prominent feature, can barely be detected by ear using a communications receiver.

10.27.4 Baseband Signal Processing

The 1000/1200-Hz pulses and 100-Hz subcarrier are first separated using a 600-Hz bandpass filter centered on 1100 Hz and a 150-Hz lowpass filter. The minute pulse is extracted using an 800-ms synchronous matched filter and pulse grooming logic which discriminates between WWV and WWVH signals and noise. The second pulse is extracted using a 5-ms FIR matched filter for each station and a single 8000-stage comb filter.

The phase of the 100-Hz subcarrier relative to the second pulse is fixed at the transmitter; however, the audio stage in many radios affects the phase response at 100 Hz in unpredictable ways. The driver adjusts for each radio using two 170-ms synchronous matched filters. The I (in-phase) filter is used to demodulate the subcarrier envelope, while the Q (quadrature-phase) filter is used in a type-1 phase-lock loop (PLL) to discipline the demodulator phase.

A bipolar data signal is determined from the matched filter subcarrier envelope using a pulse-width discriminator. The discriminator samples the I channel at 15 ms ($n$), 200 ms ($s_0$) and 500 ms ($s_1$), and the envelope (RMS I and Q channels) at 200 ms ($e_1$) and the end of the second ($e_0$). The bipolar data signal is expressed $2s_1 - s_0 - n$, where positive values correspond to data 1 and negative values correspond to data 0. Note that, since the signals $s_0$ and $s_1$ include the noise $n$, the noise component cancels out. The data bit SNR is calculated as $20 \log_{10}(e_1 / e_0)$. If the driver has not synchronized to the minute pulse, or if the data bit amplitude $e_1$ or SNR are below thresholds, the bit is considered invalid and the bipolar signal is forced to zero.

The bipolar signal is exponentially averaged in a set of 60 accumulators, one for each second, to determine the semi-static miscellaneous bits, such as DST indicator, leap second warning and DUT1 correction. In this design a data average value larger than a positive threshold is interpreted as +1 (hit) and a value smaller than a negative threshold as a -1 (miss). Values between the two thresholds, which can occur due to signal fades, are interpreted as an erasure and result in no change of indication.
10.27.5 Maximum-Likelihood Decoder

The BCD digit in each digit position of the timecode is represented as four data bits. The bits are correlated with the bits corresponding to each of the valid decimal digits in this position. If any of the four bits are invalid, the correlated value for all digits in this position is assumed zero. In either case, the values for all digits are exponentially averaged in a likelihood vector associated with this position. The digit associated with the maximum over all averaged values then becomes the maximum-likelihood candidate for this position and the ratio of the maximum over the next lower value represents the digit SNR.

The decoding matrix contains nine row vectors, one for each digit position. Each row vector includes the maximum-likelihood digit, likelihood vector and other related data. The maximum-likelihood digit for each of the nine digit positions becomes the maximum-likelihood time of the century. A built-in transition function implements a conventional clock with decimal digits that count the minutes, hours, days and years, as corrected for leap seconds and leap years. The counting operation also rotates the likelihood vector corresponding to each digit as it advances. Thus, once the clock is set, each clock digit should correspond to the maximum-likelihood digit as transmitted.

Each row of the decoding matrix also includes a compare counter and the most recently determined maximum-likelihood digit. If a digit likelihood exceeds the decision level and compares with previous digits for a number of successive minutes in any row, the maximum-likelihood digit replaces the clock digit in that row. When this condition is true for all rows and the second epoch has been reliably determined, the clock is set (or verified if it has already been set) and delivers correct time to the integral second. The fraction within the second is derived from the logical master clock, which runs at 8000 Hz and drives all system timing functions.

10.27.6 Master Clock Discipline

The logical master clock is derived from the audio codec clock. Its frequency is disciplined by a frequency-lock loop (FLL) which operates independently of the data recovery functions. The maximum value of the 5-ms pulse after the comb filter represents the on-time epoch of the second. At averaging intervals determined by the measured jitter, the frequency error is calculated as the difference between the epochs over the interval divided by the interval itself. The sample clock frequency is then corrected by this amount divided by a time constant of 8.

When first started, the frequency averaging interval is 8 seconds, in order to compensate for intrinsic codec clock frequency offsets up to 125 PPM. Under most conditions, the averaging interval doubles in stages from the initial value to 1024 s, which results in an ultimate frequency resolution of 0.125 PPM, or about 11 ms/day.

The data demodulation functions operate using the subcarrier clock, which is independent of the epoch. However, the data decoding functions are driven by the epoch. The decoder is phase-locked to the epoch in such a way that, when the clock state machine has reliably decoded the broadcast time to the second, the epoch timestamp of that second becomes a candidate to set the system clock.

The comb filter can have a long memory and is vulnerable to noise and stale data, especially when coming up after a long fade. Therefore, a candidate is considered valid only if the 5-ms signal amplitude and SNR are above thresholds. In addition, the system clock is not set until after one complete averaging interval has passed with valid candidates.

10.27.7 Station Identification

It is important that the logical clock frequency is stable and accurately determined, since in many applications the shortwave radio will be tuned to a fixed frequency where WWV or WWVH signals are not available throughout the day. In addition, in some parts of the US, especially on the west coast, signals from either or both WWV and WWVH may be available at different times or even at the same time. Since the propagation times from either station are almost always different, each station must be reliably identified before attempting to set the clock.

Reliable station identification requires accurate discrimination between very weak signals in noise and noise alone. The driver very aggressively soaks up every scrap of signal information, but has to be careful to avoid making pseudo-sense of noise alone. The signal quality metric depends on the minute pulse amplitude and SNR measured in second 0
of the minute, together with the data subcarrier amplitude and SNR measured in second 1. If all four values are above defined thresholds a hit is declared, otherwise a miss. In principle, the data pulse in second 58 is usable, but the AGC in most radios is not fast enough for a reliable measurement.

The number of hits declared in the last 6 minutes for each station represents the high order bits of the metric, while the current minute pulse amplitude represents the low order bits. Only if the metric is above a defined threshold is the station signal considered acceptable. The metric is also used by the autotune function described below and reported in the timecode string.

### 10.27.8 Performance

It is the intent of the design that the accuracy and stability of the indicated time be limited only by the characteristics of the ionospheric propagation medium. Conventional wisdom is that manual synchronization via oscilloscope and HF medium is good only to a millisecond under the best propagation conditions. The performance of the NTP daemon disciplined by this driver is clearly better than this, even under marginal conditions.

The figure below shows the measured offsets over a typical day near the bottom of the sunspot cycle ending in October, 2006. Variations up to ±0.4 ms can be expected due to changing ionospheric layer height and ray geometry over the day and night.

The figure was constructed using a 2.4-GHz P4 running FreeBSD 6.1. For these measurements the computer clock was disciplined within a few microseconds of UTC using a PPS signal and GPS receiver and the measured offsets determined from the filegen peerstats data.

The predicted propagation delay from the WWV transmitter at Boulder, CO, to the receiver at Newark, DE, varies over 9.0-9.3 ms. In addition, the receiver contributes 4.7 ms and the 600-Hz bandpass filter 0.9 ms. With these values, the mean error is less than 0.1 ms and varies ±0.3 ms over the day as the result of changing ionospheric height and ray geometry.
**10.27.9 Program Operation**

The driver begins operation immediately upon startup. It first searches for one or both of the stations WWV and WWVH and attempts to acquire minute synch. This may take some fits and starts, as the driver expects to see several consecutive minutes with good signals and low jitter. If the autotune function is active, the driver will rotate over all five frequencies and both WWV and WWVH stations until finding a station and frequency with acceptable metric.

While this is going on the the driver acquires second synch, which can take up to several minutes, depending on signal quality. When minute synch has been acquired, the driver accumulates likelihood values for the unit (seconds) digit of the nine timecode digits, plus the seven miscellaneous bits included in the WWV/H transmission format. When a good unit digit has been found, the driver accumulated likelihood values for the remaining eight digits of the timecode.

When three repetitions of all nine digits have decoded correctly, which normally takes 15 minutes with good signals, and up to 40 minutes when buried in noise, and the second synch has been acquired, the clock is set (or verified) and is selectable to discipline the system clock.

Once the clock is set, it continues to provide correct timecodes as long as the signal metric is above threshold, as described in the previous section. As long as the clock is correctly set or verified, the system clock offsets are provided once each minute to the reference clock interface, where they are processed using the same algorithms as with other reference clocks and remote servers.

It may happen as the hours progress around the clock that WWV and WWVH signals may appear alone, together or not at all. When the driver has mitigated which station and frequency is best, it sets the reference identifier to the string WVf for WWV and WHf for WWVH, where f is the frequency in megahertz. If the propagation delays have been properly set with the `fudge time1` (WWV) and `fudge time2` (WWVH) commands in the configuration file, handover from one station to the other is seamless.

Operation continues as long as the signal metric from at least one station on at least one frequency is acceptable. A consequence of this design is that, once the clock is set, the time and frequency are disciplined only by the second synch pulse and the clock digits themselves are driven by the clock state machine. If for some reason the state machine drifts to the wrong second, it would never resynchronize. To protect against this most unlikely situation, if after two days with no signals, the clock is considered unset and resumes the synchronization procedure from the beginning.

Once the system clock been set correctly it will continue to read correctly even during the holdover interval, but with increasing dispersion. Assuming the system clock frequency can be disciplined within 1 PPM, it can coast without signals for several days without exceeding the NTP step threshold of 128 ms. During such periods the root distance increases at 15 $\mu$s per second, which makes the driver appear less likely for selection as time goes on. Eventually, when the distance due all causes exceeds 1 s, it is no longer suitable for synchronization. Ordinarily, this happens after about 18 hours with no signals. The `tinker maxdist` configuration command can be used to change this value.

**10.27.10 Autotune**

The driver includes provisions to automatically tune the radio in response to changing radio propagation conditions throughout the day and night. The radio interface is compatible with the ICOM CI-V standard, which is a bidirectional serial bus operating at TTL levels. The bus can be connected to a standard serial port using a level converter such as the CT-17. Further details are on the Reference Clock Audio Drivers page.

If specified, the driver will attempt to open the device `/dev/icom` and, if successful will activate the autotune function and tune the radio to each operating frequency in turn while attempting to acquire minute synch from either WWV or WWVH. However, the driver is liberal in what it assumes of the configuration. If the `/dev/icom` link is not present or the open fails or the CI-V bus is inoperative, the driver quietly gives up with no harm done.

Once acquiring minute synch, the driver operates as described above to set the clock. However, during seconds 59, 0 and 1 of each minute it tunes the radio to one of the five broadcast frequencies to measure the signal metric as described above. Each of the five frequencies are probed in a five-minute rotation to build a database of current propagation conditions for all signals that can be heard at the time. At the end of each probe a mitigation procedure scans the database and retunes the radio to the best frequency and station found. For this to work well, the radio should
be set for a fast AGC recovery time. This is most important while tracking a strong signal, which is normally the case, and then probing another frequency, which may have much weaker signals.

The mitigation procedure selects the frequency and station with the highest valid metric, ties going first to the highest frequency and then to WWV in order. A station is considered valid only if the metric is above a specified threshold; if no station is above the metric, the rotating probes continue until a valid station is found.

The behavior of the autotune function over a typical day is shown in the figure below.

As expected, the lower frequencies prevail when the ray path is in moonlight (0100-1300 UTC) and the higher frequencies when the path is in sunlight (1300-0100 UTC). Note three periods in the figure show zero frequency when signals are below the minimum for all frequencies and stations.

### 10.27.11 Debugging Aids

The most convenient way to track the driver status is using the `ntpq` program and the `clockvar` command. This displays the last determined timecode and related status and error counters, even when the driver is not disciplining the system clock. If the debugging trace feature (`-d` on the `ntpd` command line) is enabled, the driver produces detailed status messages as it operates. If the fudge flag 4 is set, these messages are written to the `clockstats` file. All messages produced by this driver have the prefix `wwv` for convenient filtering with the Unix `grep` command.

The autotune process produces diagnostic information along with the timecode. This is very useful for evaluating the performance of the algorithms, as well as radio propagation conditions in general. The message is produced once each minute for each frequency in turn after minute synch has been acquired.

```
wwv5 status agc epoch secamp/secsnr datamp/datsnr wwv wwvh
```

where the fields after the `wwv5` identifier are: status contains status bits, agc audio gain, epoch second epoch, secamp/secsnr second pulse amplitude/SNR, and `wwv` and `wwvh` are two sets of fields, one each for WWV and WWVH. Each of the two fields has the format

```
ident score metric minamp/minsnr
```
where \textit{ident} encodes the station (WV for WWV, WH for WWVH) and frequency (2, 5, 10, 15 or 20), \textit{score} 32-bit shift register recording the hits (1) and misses (0) of the last 32 probes (hits and misses enter from the right), \textit{metric} is described above, and \textit{minamp/minsnr} is the minute pulse amplitude/SNR. An example is:

```
wwv5 000d 111 5753 3967/20.1 3523/10.2 WV20 bdef 100 8348/30.0 WH20 0000 1 22/-12.4
```

There are several other messages that can occur; these are documented in the source listing.

### 10.27.12 Monitor Data

When enabled by the filegen facility, every received timecode is written to the clockstats file in the following format:

```
sg yyyy ddd hh:mm:ss l d du lset agc ident metric errs freq avg
```

The fields beginning with \textit{yyyy} and extending through \textit{du} are decoded from the received data and are in fixed-length format. The remaining fields are in variable-length format. The fields are as follows:

- \textbf{s} The synch indicator is initially \textit{?} before the clock is set, but turns to space when all nine digits of the timecode are correctly set and the decoder is synchronized to the station within 125 \( \mu s \).

- \textbf{q} The quality character is a four-bit hexadecimal code showing which alarms have been raised. Each bit is associated with a specific alarm condition according to the following:
  - \textit{0x8} synch alarm. The decoder is not synchronized to the station within 125 \( \mu s \).
  - \textit{0x4} Digit error alarm. Less than nine decimal digits were found in the last minute.
  - \textit{0x2} Error alarm. More than 40 data bit errors were found in the last minute.
  - \textit{0x1} Compare alarm. A maximum-likelihood digit failed to agree with the current associated clock digit in the last minute.

It is important to note that one or more of the above alarms does not necessarily indicate a clock error, but only that the decoder has detected a marginal condition.

- \textbf{yyyy ddd hh:mm:ss} The timecode format itself is self explanatory. Since the driver latches the on-time epoch directly from the second synch pulse, the seconds fraction is always zero. Although the transmitted timecode includes only the year of century, the Gregorian year is augmented by 2000.

- \textbf{l} The leap second warning is normally space, but changes to \textit{L} if a leap second is to occur at the end of the month.

- \textbf{d} The DST state is \textit{S} or \textit{D} when standard time or daylight time is in effect, respectively. The state is \textit{I} or \textit{O} when daylight time is about to go into effect or out of effect, respectively.

- \textbf{du} The DUT sign and magnitude shows the current UT1 offset relative to the displayed UTC time, in deciseconds.

- \textbf{lset} Before the clock is set, the interval since last set is the number of minutes since the driver was started; after the clock is set, this is number of minutes since the decoder was last synchronized to the station within 125 \( \mu s \).

- \textbf{agc} The audio gain shows the current codec gain setting in the range 0 to 255. Ordinarily, the receiver audio gain control should be set for a value midway in this range.

- \textbf{ident} The station identifier shows the station, WVf for WWV or WHf for WWVH, and frequency ‘f’ being tracked. If neither station is heard on any frequency, the reference identifier shows \textit{NONE}.

- \textbf{metric} The signal metric described above from 0 (no signal) to 100 (best).

- \textbf{errs} The bit error counter is useful to determine the quality of the data signal received in the most recent minute. It is normal to drop a couple of data bits even under good signal conditions and increasing numbers as conditions worsen. While the decoder performs moderately well even with half the bits are in error in any minute, usually by that point the metric drops below threshold and the decoder switches to a different frequency.
freq The frequency offset is the current estimate of the codec frequency offset to within 0.1 PPM. This may wander a bit over the day due to local temperature fluctuations and propagation conditions.

avg The averaging time is the interval between frequency updates in powers of two to a maximum of 1024 s. attainment of the maximum indicates the driver is operating at the best possible resolution in time and frequency.

An example timecode is:

```
0 2000 006 22:36:00 S +3 1 115 WV20 86 5 66.4 1024
```

Here the clock has been set and no alarms are raised. The year, day and time are displayed along with no leap warning, standard time and DUT +0.3 s. The clock was set on the last minute, the AGC is safely in the middle of the range 0-255, and the receiver is tracking WWV on 20 MHz. Good receiving conditions prevail, as indicated by the metric 86 and 5 bit errors during the last minute. The current frequency is 66.4 PPM and the averaging interval is 1024 s, indicating the maximum precision available.

### 10.27.13 Fudge Factors

**time1 time** Specifies the propagation delay for WWV (40:40:49.0N 105:02:27.0W), in seconds and fraction, with default 0.0.

**time2 time** Specifies the propagation delay for WWVH (21:59:26.0N 159:46:00.0W), in seconds and fraction, with default 0.0.

**stratum number** Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid string** Ordinarily, this field specifies the driver reference identifier; however, the driver sets the reference identifier automatically as described above.

**flag1 0 | 1** Not used by this driver.

**flag2 0 | 1** Specifies the microphone port if set to zero or the line-in port if set to one. It does not seem useful to specify the compact disc player port.

**flag3 0 | 1** Enables audio monitoring of the input signal. For this purpose, the speaker volume must be set before the driver is started.

**flag4 0 | 1** Enable verbose clockstats recording if set.

### 10.28 Forum Graphic GPS Dating station

#### 10.28.1 Synopsis

Address: 127.127.37.u
Reference ID: GPS
Driver ID: GPS
Parallel Port: /dev/fgclocku

#### 10.28.2 Description

This driver supports the Forum Graphic GPS Dating station sold by EMR company.

Unfortunately sometime FG GPS start continues reporting of the same date. The only way to fix this problem is GPS power cycling and ntpd restart after GPS power-up.
After Jan 10, 2000 my FG GPS unit start send a wrong answer after 10:00am till 11:00am. It repeat hour value in result string twice. I wrote a small code to avoid such problem. Unfortunately I have no second FG GPS unit to evaluate this problem. Please let me know if your GPS has no problems after Y2K.

10.28.3 Monitor Data

Each timecode is written to the clockstats file in the format YYYY YD HH MI SS.

10.28.4 Fudge Factors

time1 time Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.
time2 time Not used by this driver.
stratum number Specifies the driver stratum, in decimal from 0 to 15, with default 0.
refid string Specifies the driver reference identifier, an ASCII string from one to four characters, with default FG.
flag1 0 | 1 Not used by this driver.
flag2 0 | 1 Not used by this driver.
flag3 0 | 1 Not used by this driver.
flag4 0 | 1 Not used by this driver.

Dmitry Smirnov (das@amt.ru)

10.29 hopf Serial Line Receivers (6021 and kompatible)

10.29.1 Synopsis

Address: 127.127.38.X
Reference ID: .hopf. (default), GPS, DCF
Driver ID: HOPF_S
Serial Port: /dev/hopfclockX
Serial I/O: 9600 baud, 8-bits, 1-stop, no parity
10.29.2 Description

The refclock_hopf_serial driver supports hopf electronic receivers with serial Interface kompatibel 6021. Additional software and information about the software drivers is available from: http://www.ATLSoft.de/ntp. Latest NTP driver source, executables and documentation is maintained at: http://www.ATLSoft.de/ntp

10.29.3 Operating System Compatibility

The hopf clock driver has been tested on the following software and hardware platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386 (PC)</td>
<td>Linux</td>
</tr>
<tr>
<td>i386 (PC)</td>
<td>Windows NT</td>
</tr>
<tr>
<td>i386 (PC)</td>
<td>Windows 2000</td>
</tr>
</tbody>
</table>

10.29.4 O/S Serial Port Configuration

The driver attempts to open the device /dev/hopfclock<X> where <X> is the NTP refclock unit number as defined by the LSB of the refclock address. Valid refclock unit numbers are 0 - 3.

The user is expected to provide a symbolic link to an available serial port device. This is typically performed by a command such as:

```bash
ln -s /dev/ttyS0 /dev/hopfclock0
```

Windows NT does not support symbolic links to device files. COMx: is used by the driver, based on the refclock unit number, where unit 1 corresponds to COM1: and unit 3 corresponds to COM3:

10.29.5 Fudge Factors

\texttt{time1 <time>} Specifies the time offset calibration factor, in seconds and fraction, with default 0.0. Should be set to 20 milliseconds to correct serial line and operating system delays incurred in capturing time stamps from the synchronous packets.

\texttt{refid <string>} Specifies the driver reference identifier, GPS or DCF.

\texttt{flag1 0 | 1} When set to 1, driver sync’s even if only crystal driven.

10.29.6 Data Format

as specified in clock manual under pt. [ Data String for NTP ( *Network Time Protocol* ) ]

Questions or Comments:

Bernd Altmeier Ing.-Büro für Software www.ATLSoft.de
10.30  *hopf* PCI-Bus Receiver (6039 GPS/DCF77)

10.30.1  Synopsis

Address: 127.127.39.X  
Reference ID: .hopf. (default), GPS, DCF  
Driver ID: HOPF_P

10.30.2  Description

The *refclock_hopf_pci* driver supports the *hopf* PCI-bus interface 6039 GPS/DCF77. Additional software and information about the software drivers maybe available from: http://www.ATLSoft.de/ntp. Latest NTP driver source, executables and documentation is maintained at: http://www.ATLSoft.de/ntp

10.30.3  Operating System Compatibility

The hopf clock driver has been tested on the following software and hardware platforms:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386 (PC)</td>
<td>Linux</td>
</tr>
<tr>
<td>i386 (PC)</td>
<td>Windows NT</td>
</tr>
<tr>
<td>i386 (PC)</td>
<td>Windows 2000</td>
</tr>
</tbody>
</table>

10.30.4  O/S System Configuration

**UNIX**

The driver attempts to open the device `:ref:'/dev/hopf6039 <driver39-REFID>'`. The device entry will be made by the installation process of the kernel module for the PCI-bus board. The driver sources belongs to the delivery equipment of the PCI-board.

**Windows NT/2000**

The driver attempts to open the device by calling the function “OpenHopfDevice()”. This function will be installed by the Device Driver for the PCI-bus board. The driver belongs to the delivery equipment of the PCI-board.

10.30.5  Fudge Factors

`refid <string>`  Specifies the driver reference identifier, GPS or DCF.

`flag1 0 | 1`  When set to 1, driver sync’s even if only crystal driven.

Questions or Comments:

Bernd Altmeier Ing.-Büro für Software www.ATLSoft.de
10.31 JJY Receivers

10.31.1 Synopsis

Address: 127.127.40.u
Reference ID: JJY
Driver ID: JJY
Serial Port: /dev/jjyu; See corresponding receiver

10.31.2 Description

This driver supports the following the JJY receivers and the GPS clock sold in Japan, and the time service through a telephone line.

<table>
<thead>
<tr>
<th>Tristate Ltd.</th>
<th>TS-JJY01, TS-JJY02</th>
<th>SEIKO TIME SYSTEMS INC.</th>
<th>TDC-300</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-DEX Co.,Ldt.</td>
<td>JST2000</td>
<td>Telephone JJY</td>
<td></td>
</tr>
<tr>
<td>Tristate Ltd.</td>
<td>TS-GPSclock-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Tristate Ltd. TS-JJY01, TS-JJY02 [http://www.tristate.ne.jp/] (Japanese only)

NTP configuration (ntp.conf)

server 127.127.40.X mode 1
fudge 127.127.40.X time1 0.0NNN flag1 011

Time1 may specify a constant to be added to the time offset for the time from the receiver, a fixed-point decimal number in seconds. You may specify the time offset from several tens of milli-seconds (0.0NNN seconds) to a hundred and several tens of milli-seconds (0.1NNN seconds) for this clock.

Flag1 has no effect for time synchronization. When flag1 is set to 1, status commands are issued before DATE and STIM commands, and write a response text into the clockstats file.

<table>
<thead>
<tr>
<th>0 (Default)</th>
<th>DCST and STUS commands are not issued</th>
<th>1</th>
<th>DCST and STUS commands are issued</th>
</tr>
</thead>
</table>

Interface RS-232C, 9600 BPS, 8-bits, no parity, 1 stop bit

Time code format

<table>
<thead>
<tr>
<th>Command</th>
<th>-&gt; Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcst{CR}{LF}</td>
<td>valid{CR}{LF}</td>
</tr>
<tr>
<td>stus{CR}{LF}</td>
<td>adjusted{CR}{LF}</td>
</tr>
<tr>
<td>time{CR}{LF}</td>
<td>HH:MM:SS{CR}{LF}</td>
</tr>
<tr>
<td>date{CR}{LF}</td>
<td>YYYY/MM/DD WWW{CR}{LF}</td>
</tr>
<tr>
<td>stim{CR}{LF}</td>
<td>HH:MM:SS{CR}{LF}</td>
</tr>
</tbody>
</table>

The date and time are requested separately. The time is requested before and after the date request to check uncertainty of the date whether it's before or after midnight.

- C-DEX Co.,Ldt. JST2000 [http://www.c-dex.co.jp/] (Japanese only)

NTP configuration (ntp.conf)
server 127.127.40.X mode 2
fudge 127.127.40.X time1 0.NNN

- Echo Keisokuki Co.,Ltd. LT-2000 [http://www.clock.co.jp/](http://www.clock.co.jp/) (Japanese only)

NTP configuration (ntp.conf)
server 127.127.40.X mode 3
fudge 127.127.40.X time1 0.NNN


NTP configuration (ntp.conf)
server 127.127.40.X mode 4
fudge 127.127.40.X time1 0.NNN

- Tristate Ltd. TS-GPSClock-01 [http://www.tristate.ne.jp/](http://www.tristate.ne.jp/) (Japanese only)

This driver supports the Tristate TS-GPSClock-01 in command/response mode, though it is a GPS clock, not JJY radio clock. Using the menus and the onboard switches, the TS-GPSClock-01 should be set to command/response mode and JST time zone.

Besides this driver (Type 40), the generic NMEA GPS driver (Type 20) supports the TS-GPSClock-01 in NMEA mode.

NTP configuration (ntp.conf)
server 127.127.40.X mode 5
fudge 127.127.40.X time1 0.NNN flag1 0|1

Time1 may specify a constant to be added to the time offset for the time from the receiver, a fixed-point decimal number in seconds.

Flag1 has no effect for time synchronization. When a flag1 is set to 1, status command is issued before DATE and TIME commands, and write a response text into a clockstats file.

<table>
<thead>
<tr>
<th>0 (Default)</th>
<th>STUS command is not issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STUS command is issued</td>
</tr>
</tbody>
</table>

Interface

USB (/dev/ttyACM*0*)

Time code format

<table>
<thead>
<tr>
<th>Command</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>stus(CR){LF}</td>
<td>*R(CR){LF}</td>
</tr>
<tr>
<td>time(CR){LF}</td>
<td>HH:MM:SS(CR){LF}</td>
</tr>
<tr>
<td>date(CR){LF}</td>
<td>YYYY/MM/DD(CR){LF}</td>
</tr>
<tr>
<td>time(CR){LF}</td>
<td>HH:MM:SS(CR){LF}</td>
</tr>
</tbody>
</table>

The date and time are requested separately. The time is requested before and after the date request to check uncertainty of the date whether it’s before or after midnight.

10.31. JJY Receivers 247
The TDC-300 must be set to the type 3 data format using the front panel menu display and the switches.

NTP configuration (ntp.conf)

```plaintext
server 127.127.40.X mode 6
fudge 127.127.40.X time1 0.NNN
```

Interface

RS-232C, 2400 BPS, 8-bits, no parity, 1 stop bit

Time code format

<table>
<thead>
<tr>
<th>Command</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXYYMMDDWHHMMSSETX</td>
<td>STXxE5ETX</td>
</tr>
<tr>
<td>(5 to 10 mSec. before second)</td>
<td></td>
</tr>
</tbody>
</table>

The telephone JJY is the time service through a public telephone line. The service is provided by the National Institute of Information and Communications Technology in Japan.

ATTENTION: This mode, the telephone JJY, can not be used with the refclock Acts (type 18) at the same time. Because the “phone” statement in the ntp configuration file is not involved with the “server” statement, so the both the refclock Acts (type 18) and this refclock JJY (type 40, mode 100 to 180) can not recognize the appropriate “phone” statement among the “phone” statements.

NTP configuration (ntp.conf)

```plaintext
server 127.127.40.X mode (100, 101 to 180) minpoll N
```

The mode 100 is specified, this driver does not issue the loopback command in order to measure the delay, and the delay of the telephone line and the system processing is not adjusted. The mode 101 to 180 is specified, this driver issues the loopback command and measures the delay of the telephone line and the system processing through the Telephone JJY loopback circuit. The round trip time through the Telephone JJY loopback circuit is measured 5 times, and each delay time is greater than 700 milli-seconds, that delay time is ignored during average delay time calculation. Also, if the valid delay time (< 700 mS.) is measured more than 3 times, the maximum delay time among the valid delay times is ignored, and if the valid delay time is measured more than 4 times, the minimum delay time among them is ignored, like marking/grading sports judgment. The adjustment time is calculated by the formula, multiply (the measured round trip time) by ( (the mode number) - 100) %, and the adjustment delay time is added to the synchronizing time. If you choose the automatic delay adjustment, in other words, the mode 101 to 180 is specified, the recommended mode number is 145 to 165.

The default polling interval 6 (64 seconds) is too short for this mode. The “minpoll” should be set to greater than or equal to 8 (256 seconds, about 4 minutes). The interval time is given the value in second power of 2. The minpoll value 12 is 4096 seconds interval (about 1 hour), 14 is 16384 seconds interval (about 4.5 hours), 16 is 65536 seconds (about 18 hours), respectively.

```plaintext
fudge 127.127.40.X flag1 011 flag2 011 flag3 011 flag4 011
```

Time1 may specify a constant to be added to the time offset for the time from the receiver, a fixed-point decimal number in seconds. When the mode 100 is specified, the time1 may be specified in order to adjust the time offset. When the mode 101 to 180 is specified, the time1 should not be specified because this driver adds some percentage of the measured loopback delay, depending on the value of the mode number.
Flag1 is the modem dialing type.

<table>
<thead>
<tr>
<th>Flag1</th>
<th>Dialing Type</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>Tone</td>
<td>ATDWTnn...nn</td>
</tr>
<tr>
<td>1</td>
<td>Pulse</td>
<td>ATDWPnn...nn</td>
</tr>
</tbody>
</table>

Flag2 is the modem error correction type.

<table>
<thead>
<tr>
<th>Flag2</th>
<th>Error Correction Type</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>Normal</td>
<td>AT\N0</td>
</tr>
<tr>
<td>1</td>
<td>Auto V42, MNP, Normal</td>
<td>AT\N3</td>
</tr>
</tbody>
</table>

Flag3 is the modem speaker switch.

<table>
<thead>
<tr>
<th>Flag3</th>
<th>Speaker Switch</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>Off</td>
<td>ATM0Ln</td>
</tr>
<tr>
<td>1</td>
<td>On</td>
<td>ATM2Ln</td>
</tr>
</tbody>
</table>

Flag4 is the modem speaker volume.

<table>
<thead>
<tr>
<th>Flag4</th>
<th>Volume</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Default)</td>
<td>Low</td>
<td>ATMnL1</td>
</tr>
<tr>
<td>1</td>
<td>Middle</td>
<td>ATMnL2</td>
</tr>
</tbody>
</table>

phone 042NNNNNNNN

The phone number is available at [http://jjy.nict.go.jp/](http://jjy.nict.go.jp/)

The number of digits of the phone number is checked. If the international access number and the country number are added in order to call from outside of Japan, the number of digits is over the limit.

The first 2 or 3 digits are checked. The emergency service number and the special service number in Japan are not allowed.

Calling from extension line, the number for an outside line should be prefix “0,” ( Zero, Comma ). The prefix is also checked, and no other outside access number is allowed.

Interface

RS-232C or USB, 2400 BPS, 8-bits, no parity, 1 stop bit

Modem control commands:

ATE0Q0V1, ATMnLn, AT&K4, AT+MS=V22B, AT%C0, AT\Nn, ATH1, ATDWxnn...nn

+++ , ATH0

Time code format

10.31. JJY Receivers 249
The date and time are requested separately. The time is requested before and after the date request to check uncertainty of the date whether it’s before or after midnight.

The leap second is not handled, and only written in the clockstats file as an information.

The JJY is the radio station which transmits the JST (Japan Standard Time) in long wave radio. The station JJY is operated by the National Institute of Information and Communications Technology. An operating announcement and some information are available from http://www.nict.go.jp/ (English and Japanese) and http://jjy.nict.go.jp/ (English and Japanese)

The user is expected to provide a symbolic link to an available serial port device. This is typically performed by a command such as:

```
ln -s /dev/ttyS0 /dev/jjy0
```

Using an RS-232C to USB converter cable, the clock or a modem can be connected to a USB port instead of a serial port. In this case, the typical symbolic link command is as follows;

```
ln -s /dev/ttyUSB0 /dev/jjy0
```

Windows NT does not support symbolic links to device files. COMX: is the unit used by the driver, based on the refclock unit number, where unit 1 corresponds to COM1: and unit 3 corresponds to COM3:

### 10.31.3 Monitor Data

The driver writes sent and received data to/from the JJY receivers, GPS clock, and the modem into the `clockstats` file.

```
statsdir /var/log/ntpd/
filegen clockstats file clockstats type day enable
```

Mark of the clockstats record
10.31.4 Fudge Factors

**time1**  *time*  Specifies the time offset calibration factor, in seconds and fraction, with default 0.0.

**time2**  *time*  Not used by this driver.

**stratum**  *number*  Specifies the driver stratum, in decimal from 0 to 15, with default 0.

**refid**  *string*  Specifies the driver reference identifier, an ASCII string from one to four characters, with default JJY.

**flag1**  0 | 1  See corresponding receiver.

**flag2**  0 | 1  See corresponding receiver.

**flag3**  0 | 1  See corresponding receiver.

**flag4**  0 | 1  See corresponding receiver.

10.32 Zyfer GPStarplus Receiver

10.32.1 Synopsis

Address: 127.127.42.u
Reference ID: GPS
Driver ID: Zyfer GPStarplus
Serial Port: /dev/zyferu; 9600 baud, 8-bits, no parity
Features: (none)

10.32.2 Description

This driver supports the Zyfer GPStarplus receiver.

The receiver has a DB15 port on the back which has input TxD and RxD lines for configuration and control, and a separate TxD line for the once-per-second timestamp.

Additionally, there are BNC connectors on the back for things like PPS and IRIG output.

10.32.3 Additional Information

Reference Clock Drivers
10.33 RIPE NCC interface for Trimble Palisade

10.33.1 Synopsis

Address: 127.127.43.u
Reference ID: RIPE-NCC
Driver ID: RIPE-NCC

10.33.2 Description

This is a special driver developed to be used in conjunction with the RIPE NCC clock card in the RIPE NCC Test Traffic Measurements project.

10.33.3 Why this driver?

The reason why we created a separated driver for an antenna for which already a (vendor supplied) driver exist is a design decision. To be more specific, the standard Trimble interface uses a 12 pin connector. The cable sold by Trimble to connect to this wire is a very thick cable. Certainly not something you wish to run for several 100 meters through your building. And if you wanted to run it for 100 meters, you always would have to really run the cable, and didn’t have the option to use existing wiring.

This is where we wanted more flexibility. We wanted to be able to use existing wiring in buildings. That leaded us to CAT-5(UTP) which only gives us 8 wires. Therefore we decided to redesign the use of the Trimble antenna. The Trimble supports two modes: EVENT driver and PPS mode. The default is to use the EVENT mode which needs all 12 wires. We only use the PPS timestamps for which we have enough with 8 wires. For our purposes this is more than fine.

More information about the project can be found on the Test Traffic Measurements website.
10.33.4 RIPE NCC clock card

The card is very a simple PCI card. The only feature on the bus it uses is the power supply. It uses this power supply to power the Trimble GPS antenna.

The card basically just is a RS422 to RS232 converter. It gets the Trimble’s RS422 signal on a RJ45 connector and transforms that to RS232 on a DIN9 connector. This connector should be loopbacked on the back of the machine to the serial port. As said, the card doesn’t do any PCI data transfers.

The schematics of the interface card is available here: gps_interface_schematic.pdf. You are free to create this card yourself as long as you give some credit or reference to us. Note that we don’t sell these cards on a commercial basis, but for interested parties we do have some spares to share.

10.33.5 Monitor Data

In the filegen clockstats file the following (example) data is collected:

```
52445 41931.275 127.127.40.0 U1 20.6.2002 11:38:51 13 11
52445 41931.395 127.127.40.0 C1 20062002 113851 6 364785 110.2 450 6.7 13 5222.374737
52445 41931.465 127.127.40.0 S1 07 1 1 02 59.3 291.5 39.3
52445 41931.485 127.127.40.0 S1 11 2 1 02 59.9 138.0 60.2
52445 41931.525 127.127.40.0 S1 01 4 1 02 48.4 185.7 28.3
52445 41931.555 127.127.40.0 S1 14 5 2 02 32.7 41.0 15.4
52445 41931.585 127.127.40.0 S1 20 6 1 02 59.9 256.6 78.0
52445 41931.615 127.127.40.0 S1 25 8 2 00 0.0 86.6 20.1
```

This is in the form of:

All output lines consist of a prefix and a message, the prefix is:
[days since epoch] [sec.ms since start of day] [peer address]

And all individual messages:

*Primary UTC time packet:
U1 [date] [time] [trackstat] [utcflags]

*Comprehensive time packet:
C1 [date] [time] [mode] [bias] [biasunc] [rate] [rateunc] [utcoff] [latitude] [longtitude] [alt] [vis]

*Tracking status packet:
S1 [prn] [channel] [aqflag] [ephstat] [snr] [azinuth] [elevation]
10.33.6 Additional Information

Reference Clock Drivers

10.34 NeoClock4X - DCF77 / TDF serial line receiver

10.34.1 Synopsis

Address: 127.127.44.
Reference ID: neol
Driver ID: NEOCLK4X
Serial Port: /dev/neoclock4x-

10.34.2 Description

The refclock_neoclock4x driver supports the NeoClock4X receiver available from Linum Software GmbH. The receiver is available as a DCF77 or TDF receiver. Both receivers have the same output string. For more information about the NeoClock4X receiver please visit http://www.linux-funkuhr.de.

10.34.3 Fudge Factors

time1 time

Specifies the time offset calibration factor with the default value off 0.16958333 seconds. This offset is used to correct serial line and operating system delays incurred in capturing time stamps. If you want to fudge the time1 offset ALWAYS add a value off 0.16958333. This is necessary to compensate to delay that is caused by transmit the timestamp at 2400 Baud. If you want to compensate the delay that the DCF77 or TDF radio signal takes to travel to your site simply add the needed millisecond delay to the given value. Note that the time here is given in seconds. Default setting is 0.16958333 seconds.

time2 time

Not used by this driver.

flag1 0 | 1
When set to 1 the driver will feed ntp with timestamp even if the radio signal is lost. In this case an
ternal backup clock generates the timestamps. This is ok as long as the receiver is synced once since the
receiver is able to keep time for a long period. Default setting is 0 = don’t synchronize to CMOS clock.

flag2 0 | 1

You can allow the NeoClock4X driver to use the quartz clock even if it is never synchronized to a radio
clock. This is usually not a good idea if you want precise timestamps since the CMOS clock is maybe not
adjusted to a dst status change. So **PLEASE** switch this only on if you now what you’re doing. Default
setting is 0 = don’t synchronize to unsynchronized CMOS clock.

flag3 0 | 1

Not used by this driver.

flag4 0 | 1

It is recommended to allow extensive logging while you setup the NeoClock4X receiver. If you activate
flag4 every received data is logged. You should turn off flag4 as soon as the clock works as expected to
reduce logfile cluttering. Default setting is 0 = don’t log received data and converted utc time.

Please send any comments or question to neoclock4x@linum.com.

### 10.35 Spectracom TSYNC PCI

#### 10.35.1 Synopsis

Address: 127.127.45.u

Reference ID: one of GPS, IRIG, HVQ, FREQ, ACTS, PPS, PTP, ACT, USR, LOCL

Driver ID: Spectracom TSYNC PCI

Driver Port: /dev/tsyncpci

**Features:** (none)

#### 10.35.2 Description

This driver supports the Spectracom TSYNC PCI receiver.

#### 10.35.3 Additional Information

Reference Clock Drivers

### 10.36 GPSD NG client driver

#### 10.36.1 Synopsis

Address: 127.127.46.u

Reference ID: GPSD

Driver ID: GPSD_JSON

Serial Port: /dev/gpsu as symlink to the true device (not used directly; see below)
10.36.2 Description

This driver is a client driver to the GPSD daemon, which over the time became increasingly popular for UN*Xish platforms. GPSD can manage several devices in parallel, aggregate information, and acts as a data hub for client applications. GPSD can also auto-detect and handle PPS hardware signals on serial ports. Have a look at the GPSD project page.

It is important to understand that this driver works best using a GPS device with PPS support.

The GPSD-NG protocol is text based, using JSON notation to transfer records in form of JSON objects. The driver uses a TCP/IP connection to localhost:gpsd to connect to the daemon and then requests the GPS device /dev/gpsu to be watched. (Different clock units use different devices, and GPSD is able to give only the relevant information to a clock instance.)

This driver does not expect GPSD to be running or the clock device to be present a priori; it will try to re-establish a lost or hitherto unsuccessful connection and will wait for device to come up in GPSD. There is an initial 10 seconds delay between a connection loss or failed attempt and the next reconnect attempt; this makes sure that there is no thrashing on the network layer. If the connection fails again, an exponential back off is used with an upper limit of approximately 10 minutes.

The overall accuracy depends on the receiver used. The driver uses the error estimations (95% probability limits) provided by GPSD to set the clock precision dynamically according to these readings.

The driver needs the VERSION, TPV, PPS, WATCH and TOFF objects of the GPSD protocol. (Others are quietly ignored.) The driver can operate without the TOFF objects, which are available with the protocol version 3.10 and above. (Not to be confused with the release version of GPSD!) Running without TOFF objects has a negative impact on the jitter and offset of the serial timing information; if possible, a version of GPSD with support for TOFF objects should be used.

The acronym STI is used here as a synonym for serial time information from the data channel of the receiver, no matter what objects were used to obtain it.

10.36.3 Naming a Device

The GPSD driver uses the same device name as the NMEA driver, namely /dev/gpsu. There is a simple reason for that: While the NMEA driver and the GPSD driver can be active at the same time for different devices, they cannot access the same device at a time. Having the same name helps on that. It also eases migration from using NMEA directly to using GPSD, as no new links etc need to be created.

GPSD is normally started with the device name to access; it can also be instructed by hot-plug scripts to add or remove devices from its device pool. Luckily, the symlinks used by the NMEA driver are happily accepted and used by GPSD; this makes it possible to use the symlink names as device identification. This makes the migration from the built-in NMEA driver a bit easier.

Note: GPSD (as of version 3.10) cannot use kernel mode PPS on devices that are hot-plugged. This would require to attach the PPS line discipline to the character special file, which is not possible when running with root privileges already dropped. This is not likely to change in the future.

10.36.4 The ‘mode’ word

A few operation modes can be selected with the mode word.
<table>
<thead>
<tr>
<th>Bits</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0..1    | 0     | STI only operation. This mode is affected by the timing stability of whatever protocol is used between the GPS device and GPSD. Running on STI only is not recommended in general. Possible use cases include:  
• The receiver does not provide a PPS signal.  
• The receiver *does* provide a PPS signal and the secondary PPS unit is used.  
• The receiver has a stable serial timing and a proper fudge can be established.  
• You have other time sources available and want to establish a useful fudge value for time2. |
| 1       | 1     | Strict operation. This mode needs a valid PPS and a valid STI to combine the absolute time from the STI with the time stamp from the PPS record. Does not feed clock samples if no valid PPS+STI pair is available. This type of operation results in an ordinary clock with a very low jitter as long as the PPS data is available, but the clock fails once PPS drops out. This mode is a possible choice for receivers that provide a PPS signal most of the time but have an unstable serial timing that cannot be fudge-compensated. |
| 2       | 2     | Automatic mode. Tries to operate in strict mode unless it fails to process valid samples for some time, currently 120s. Then it reverts to STI-only operation until the PPS is stable again for 40s, when strict mode is engaged again. **Important Notice:** This is an experimental feature! Switching between strict and STI-only mode will cause changes in offset and jitter. Use this mode only if STI-only works fairly well with your setup, or if you expect longer dropouts of the PPS signal and prefer to use STI alone over not getting synchronised at all. |
|         | 3     | *(reserved for future extension, do not use)*                                                                                               |
| 2..31   |       | *(reserved for future extension, do not use)*                                                                                               |
10.36.5 Syslog flood throttle

This driver can create a lot of syslog messages when things go wrong, and cluttering the log files is frowned upon. So we attempt to log persistent or recurring errors only once per hour. On the other hand, when tracking a problem the syslog flood throttle can get into the way.

Therefore, fudge flag3 can be used to disable the flood throttle at any time; the throttle is engaged by default. Running with the syslog flood throttle disabled for lengthy time is not recommended unless the log files are closely monitored.

10.36.6 PPS secondary clock unit

Units with numbers \( \geq 128 \) act as secondary clock unit for the primary clock unit \((u \mod 128)\). A secondary unit processes only the PPS data from GPSD and needs the corresponding master unit to work \(^2\). Use the ‘noselect’ keyword on the primary unit if you are not interested in its data.

The secondary unit employs the usual precautions before feeding clock samples:

- The system must be already in a synchronised state.
- The system offset must be less than 400ms absolute.
- The phase adjustment from the PPS signal must also be less than 400ms absolute.

If fudge flag flag1 is set for the secondary unit, the unit asserts the PPS flag on the clock as long as PPS data is available. This makes the unit eligible as PPS peer and should only be used if the GPS receiver can be trusted for the quality of its PPS signal \(^3\). The PPS flag gets cleared if no PPS records can be acquired for some time. The unit also flushes the sample buffer at this point to avoid the use of stale PPS data.

**Attention:** This unit uses its own PPS fudge value which must be set as fudge time1. Only the fudge values time1 and flag1 have an impact on secondary units.

10.36.7 Clockstats

If flag4 is set when the driver is polled, a clockstats record is written for the primary clock unit. (The secondary PPS unit does not provide clock stats on its own.) The first 3 fields are the normal date, time, and IP address common to all clockstats records.

\(^2\) Data transmission an decoding is done only once by the primary unit. The decoded data is then processed independently in both clock units. This avoids double transmission over two sockets and decoding the same data twice, but the primary unit is always needed as a downside of this approach.

\(^3\) The clock driver suppresses the processing PPS records when the TPV/TIME data indicates the receiver has no fix. It can also deal with situations where the PPS signal is not delivered to GPSD. But once it is available, it is also processed and used to create samples. If a receiver cannot be trusted for the precision of its PPS signal, it should not be used to create a possible PPS peer: These get extra clout and can effectively become the sole source of input for the control loop. You do not want to use sloppy data for that.
The Clockstats Line

<table>
<thead>
<tr>
<th>The Clockstats Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td>Date as day number since NTP epoch.</td>
</tr>
<tr>
<td>2</td>
<td>Time as seconds since midnight.</td>
</tr>
<tr>
<td>3</td>
<td>(Pseudo-) IP address of clock unit.</td>
</tr>
<tr>
<td>4</td>
<td>Number of received known JSON records since last poll. The driver knows about TPV, PPS, TOFF, VERSION and WATCH records; others are silently ignored.</td>
</tr>
<tr>
<td>5</td>
<td>Bad replies since last poll. A record is considered malformed or a bad reply when it is missing vital fields or the fields contain malformed data that cannot be parsed.</td>
</tr>
<tr>
<td>6</td>
<td>Number of sample cycles since last poll that were discarded because there was no GPS fix. This is effectively the number of TPV records with a fix value &lt; 2 or without a time stamp.</td>
</tr>
<tr>
<td>7</td>
<td>Number of serial time information records (TPV or TOFF, depending on the GPSD version) received since last poll.</td>
</tr>
<tr>
<td>8</td>
<td>Number of serial time information records used for clock samples since the last poll.</td>
</tr>
<tr>
<td>9</td>
<td>Number of PPS records received since the last poll.</td>
</tr>
<tr>
<td>10</td>
<td>Number of PPS records used for clock samples on the secondary channel since the last poll.</td>
</tr>
</tbody>
</table>

10.36.8 Fudge Factors

time1 time Specifies the PPS time offset calibration factor, in seconds and fraction, with default 0.0.
time2 time [Primary Unit] Specifies the TPV/TIME time offset calibration factor, in seconds and fraction, with default 0.0.
stratum number Specifies the driver stratum, in decimal from 0 to 15, with default 0.
refid string Specifies the driver reference identifier, an ASCII string from one to four characters, with default GPSD.
flag1 0 | 1 [Secondary Unit] When set, flags the secondary clock unit as a potential PPS peer as long as good PPS data is available.
flag2 0 | 1 [Primary Unit] When set, disables the processing of incoming PPS records. Intended as an aide to test the effects of a PPS dropout when using automatic mode (mode 2).
flag3 0 | 1 [Primary Unit] If set, disables the log throttle. Useful when tracking problems in the interaction between GPSD and NTPD, since now all error events are logged. Persistent/recurrent errors can easily fill up the log, so this should only be enabled during bug hunts.
flag4 0 | 1 [Primary Unit] If set, write a clock stats line on every poll cycle.

10.36.9 Additional Information

Reference Clock Drivers
10.37 MX4200 Receiver Data Format

10.37.1 Control Port Sentences

The Control (CDU) Port is used to initialize, monitor, and control the receiver. The structure of the control port sentences is based on the NMEA-0183 Standard for Interfacing Marine Electronics Navigation Devices (version 1.5). For more details, please refer to the NMEA-0183 Specification available from the National Marine Electronics Association.

Reserved characters are used to indicate the beginning and the end of records in the data stream, and to delimit data fields within a sentence. Only printable ASCII characters (Hex 20 through 7F) may be used in a sentence. Table 2 lists the reserved characters and defines their usage. Table 1 illustrates the general Magnavox proprietary NMEA sentence format.

Table 1. Magnavox Proprietary NMEA Sentence Format

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Sentence Start Character</td>
</tr>
<tr>
<td>P</td>
<td>Special ID (P = Proprietary)</td>
</tr>
<tr>
<td>MVX</td>
<td>Originator ID (MVX = Magnavox)</td>
</tr>
<tr>
<td>G</td>
<td>Interface ID (G = GPS)</td>
</tr>
<tr>
<td>XXX</td>
<td>Sentence Type</td>
</tr>
<tr>
<td>...</td>
<td>Data</td>
</tr>
<tr>
<td>*</td>
<td>Optional Checksum Field Delimiter</td>
</tr>
<tr>
<td>CK</td>
<td>Optional Checksum</td>
</tr>
</tbody>
</table>

Following the start character $, are five characters which constitute the block label of the sentence. For Magnavox proprietary sentences, this label is always $PMVXG$. The next field after the block label is the sentence type, consisting of three decimal digits.

The data, delimited by commas, follows the sentence type. Note that the receiver uses a free-format parsing algorithm, so you need not send the exact number of characters shown in the examples. You will need to use the commas to determine how many bytes of data need to be retrieved.

The notation CK shown in Table 1 symbolically indicates the optional checksum in the examples. The checksum is computed by exclusive-ORing all of the bytes between the $ and the * characters. The $, * and the checksum are not included in the checksum computation.

Checksums are optional for Control Port input sentences, but are highly recommended to limit the effects of communication errors. Magnavox receivers always generate checksums for Control Port output sentences.

ASCII data characters are transmitted in the following format:
NULL fields are fields which do not contain any data. They would appear as two commas together in the sentence format, except for the final field. Some Magnavox proprietary sentences require that the format contain NULL fields. Mandatory NULL fields are identified by an ‘*’ next to the respective field.

### 10.37.2 Control Port Input Sentences

These are the subset of the MX4200 control port input sentences sent by the NTP driver to the GPS receiver.

#### $PMVXG,000$

**Initialization/Mode Control - Part A**

Initializes the time, position and antenna height of the MX4200.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Day</td>
<td>Int</td>
<td></td>
<td>1-31</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Month</td>
<td>Int</td>
<td></td>
<td>1-12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Year</td>
<td>Int</td>
<td></td>
<td>1991-9999</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>GMT Time</td>
<td>HHMMSS</td>
<td>Int</td>
<td>000000-235959</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WGS-84 Latitude</td>
<td>DDMM.MMMM</td>
<td>Float</td>
<td>0.0</td>
<td>0 - 8959.9999</td>
</tr>
<tr>
<td>6</td>
<td>North/South Indicator</td>
<td>Char</td>
<td></td>
<td>N</td>
<td>N,S</td>
</tr>
<tr>
<td>7</td>
<td>WGS-84 Longitude</td>
<td>DDDMM.MMMM</td>
<td>Float</td>
<td>0.0</td>
<td>0 - 17959.9999</td>
</tr>
<tr>
<td>8</td>
<td>East/West Indicator</td>
<td>Char</td>
<td></td>
<td>E</td>
<td>E,W</td>
</tr>
<tr>
<td>9</td>
<td>Altitude (height above Mean Sea Level)</td>
<td>Meters</td>
<td>Float</td>
<td>0.0</td>
<td>+/-99999.0</td>
</tr>
<tr>
<td>10</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

$PMVXG,000""""""\times48$

$PMVXG,000"\times,5128.4651,N,00020.0715,W,58.04,\times4F$

#### $PMVXG,001$

**Initialization/Mode Control - Part B**

Specifies various navigation parameters: Altitude aiding, acceleration DOP limits, and satellite elevation limits.
### Field Description Units Format Default Range

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constrain Altitude</td>
<td>Int</td>
<td>1</td>
<td>0=3D Only 1=Auto 2=2D Only</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Horizontal Acceleration Factor m/sec^2</td>
<td>Float</td>
<td>1.0</td>
<td>0.5-10.0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VDOP Limit</td>
<td>Int</td>
<td>10</td>
<td>1.9999</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>HDOP Limit</td>
<td>Int</td>
<td>10</td>
<td>1.9999</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Elevation Limit</td>
<td>Deg</td>
<td>Int</td>
<td>5</td>
<td>0-90</td>
</tr>
<tr>
<td>8</td>
<td>Time Output Mode</td>
<td>Char</td>
<td>U</td>
<td>U=UTC L=Local Time</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Local Time Offset</td>
<td>HHMM</td>
<td>Int</td>
<td>0</td>
<td>+/- 0-2359</td>
</tr>
</tbody>
</table>

Example:

$PMVXG,001,3",0.1,0.1,10,10,5,U,0*06$

$PMVXG,007$

**Control Port Output Configuration**

This message enables or disables output of the specified sentence and defines the output rate. The user sends this message for each sentence that the receiver is to output.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Control Port Output Block Label</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Clear Current Output List</td>
<td>Int</td>
<td></td>
<td>0=No 1=Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Add/Delete Sentence from List</td>
<td>Int</td>
<td></td>
<td>1=Append 2=Delete</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Sentence Output Rate</td>
<td>Sec</td>
<td>Int</td>
<td></td>
<td>1.9999</td>
</tr>
<tr>
<td>6</td>
<td># digits of Precision for CGA and GLL sentences</td>
<td>Int</td>
<td>2</td>
<td></td>
<td>2-4</td>
</tr>
<tr>
<td>7</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

$PMVXG,007,022,0,1",1",*4F$

$PMVXG,023$

**Time Recovery Configuration**

This message is used to enable/disable the time recovery feature of the receiver. The time synchronization for the 1PPS output is specified in addition to a user time bias and an error tolerance for a valid pulse. This record is accepted in units configured for time recovery. If the back panel contains a 1PPS outlet, the receiver is a time recovery unit.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>*1</td>
<td>Time Recovery Mode</td>
<td>Char</td>
<td>D</td>
<td>D</td>
<td>D=Dynamic S=Static K=Known Position N=No Time Recovery</td>
</tr>
<tr>
<td>2</td>
<td>Time Synchronization</td>
<td>Char</td>
<td>G</td>
<td>U</td>
<td>U=UTC G=GPS</td>
</tr>
<tr>
<td>3</td>
<td>Time Mark Mode</td>
<td>Char</td>
<td>A</td>
<td>A</td>
<td>A=Always V=Valid Pulses Only</td>
</tr>
<tr>
<td>4</td>
<td>Maximum Time Error</td>
<td>Nsec</td>
<td>Int</td>
<td>100</td>
<td>50-1000</td>
</tr>
<tr>
<td>5</td>
<td>User Time Bias</td>
<td>Nsec</td>
<td>Int</td>
<td>0</td>
<td>+/- 99999</td>
</tr>
<tr>
<td>6</td>
<td>ASCII Time Message Control</td>
<td>Int</td>
<td>0</td>
<td>0=No Output 1=830 to Control Port 2=830 to Equipment Port</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Known Pos PRN</td>
<td>Int</td>
<td>0</td>
<td>1-32 0=Track All Sats</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
$PMVXG,023,S,U,A,500,0,1,*16
```

**$CDGPQ,YYY**

**Query From a Remote Device / Request to Output a Sentence**

Enables the controller to request a one-time transmission of a specific block label. To output messages at a periodic rate, refer to input sentence `$PMVXG,007`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:CD</td>
<td>ID of Remote Device</td>
<td>Char</td>
<td></td>
<td>(See NMEA-0183)</td>
<td></td>
</tr>
<tr>
<td>2:GP</td>
<td>GPS</td>
<td>Char</td>
<td></td>
<td>(See NMEA-0183)</td>
<td></td>
</tr>
<tr>
<td>3:Q</td>
<td>Query</td>
<td>Char</td>
<td></td>
<td>(See NMEA-0183)</td>
<td></td>
</tr>
<tr>
<td>4:YYY</td>
<td>Label of Desired Sentence</td>
<td>Char</td>
<td></td>
<td>Any Valid NMEA or Magnavox Sentence Type</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
$CDGPQ,030*5E
```

**10.37.3 Control Port Output Sentences**

These are the subset of the MX4200 control port output sentences recognized by the NTP driver.

**$PMVXG,000**

**Receiver Status**

Returns the current status of the receiver including the operating mode, number of satellites visible, and the number of satellites being tracked.
### Field Description

<table>
<thead>
<tr>
<th>Field Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Current Receiver Status</td>
<td>Char</td>
<td></td>
<td>ACQ=Reacquisition ALT=Constellation Selection IAC=Initial Acquisition IDL=Idle, No Satellites NAV=Navigating STS=Search The Sky TRK=Tracking</td>
</tr>
<tr>
<td>2 Number of Satellites that should be Visible</td>
<td>Int</td>
<td>0-12</td>
<td></td>
</tr>
<tr>
<td>3 Number of Satellites being Tracked</td>
<td>Int</td>
<td>0-12</td>
<td></td>
</tr>
<tr>
<td>4 Time since Last Navigation</td>
<td>HHMM</td>
<td>0-2359</td>
<td></td>
</tr>
<tr>
<td>5 Initialization Status</td>
<td>Int</td>
<td>0=Waiting for Initialization 1=Initialization on Complete</td>
<td></td>
</tr>
</tbody>
</table>

Example:

$PMVXG,000,TRK,3,3,0122,1*19$

**$PMVXG.021**

**Position, Height, Velocity**

This sentence gives the receiver position, height, navigation mode and velocity north/east. *This sentence is intended for post analysis applications.*

<table>
<thead>
<tr>
<th>Field Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 UTC Measurement Time</td>
<td>Seconds into the week</td>
<td>Float</td>
<td>0-604800.00</td>
</tr>
<tr>
<td>2 WGS-84 Latitude</td>
<td>DDMM.MM</td>
<td>MMD</td>
<td>0-89.9999</td>
</tr>
<tr>
<td>3 North/South Indicator</td>
<td>Char</td>
<td></td>
<td>N, S</td>
</tr>
<tr>
<td>4 WGS-84 Longitude</td>
<td>DDDMM.MM</td>
<td>MMD</td>
<td>0-179.9999</td>
</tr>
<tr>
<td>5 East/West Indicator</td>
<td>Char</td>
<td></td>
<td>E, W</td>
</tr>
<tr>
<td>6 Altitude (MSL)</td>
<td>Meters</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>7 Geoidal Height</td>
<td>Meters</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>8 Velocity East</td>
<td>M/Sec</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>9 Velocity North</td>
<td>M/Sec</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>10 Navigation Mode</td>
<td>Int</td>
<td></td>
<td>Navigating 1=Position From a Remote Device 2=2D 3=3D 4=2D differential 5=3D differential Not Navigating 51=Too Few Satellites 52=DOPs too large 53=Position STD too large 54=Velocity STD too large 55=Too many iterations for velocity 56=Too many iterations for position 57=3 Sat Startup failed</td>
</tr>
</tbody>
</table>
Example:

$PMVXG,021,142244.00,5128.4744,N,00020.0593,W,00054.4,0047.4,0000.1,-000.2,03*66

$PMVXG,022

**DOPs**

This sentence reports the DOP (Dilution Of Precision) values actually used in the measurement processing corresponding to the satellites listed. The satellites are listed in receiver channel order. Fields 11-16 are output only on 12-channel receivers.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UTC Measurement Time</td>
<td>Seconds into the week</td>
<td>Float</td>
<td>0-604800.00</td>
</tr>
<tr>
<td>2</td>
<td>East DOP (EDOP)</td>
<td>Float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>North DOP (NDOP)</td>
<td>Float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Vertical DOP (VDOP)</td>
<td>Float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PRN on Channel #1</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PRN on Channel #2</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PRN on Channel #3</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PRN on Channel #4</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PRN on Channel #5</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>PRN on Channel #6</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>PRN on Channel #7</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PRN on Channel #8</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>PRN on Channel #9</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PRN on Channel #10</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>PRN on Channel #11</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>PRN on Channel #12</td>
<td>Int</td>
<td>1-32</td>
<td></td>
</tr>
</tbody>
</table>

Example:

$PMVXG,022,142243.00,00.7,00.8,01.9,27,26,10,09,13,23*77

$PMVXG,030

**Software Configuration**

This sentence contains the navigation processor and baseband firmware version numbers.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nav Processor Version Number</td>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Baseband Firmware Version Number</td>
<td>Char</td>
<td></td>
</tr>
</tbody>
</table>

Example:

$PMVXG,030,DA35,015

$PMVXG,101

**Control Sentence Accept/Reject**

This sentence is returned (on the Control Port) for every $PMVXG and $XXGPQ sentence that is received.
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sentence ID</td>
<td>Char</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Accept/Reject Status</td>
<td>Int</td>
<td></td>
<td>0=Sentence Accepted 1=Bad Checksum 2=Illegal Value 3=Unrecognized ID 4=Wrong # of fields 5=Required Data Field Missing 6=Requested Sentence Unavailable</td>
</tr>
<tr>
<td>3</td>
<td>Bad Field Index</td>
<td>Int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Requested Sentence ID (If field #1 = GPQ)</td>
<td>Char</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

$PMVXG,101,GPQ,0"030*0D

### $PMVXG,523

#### Time Recovery Configuration

This sentence contains the configuration of the time recovery function of the receiver.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time Recovery Mode</td>
<td>Char</td>
<td></td>
<td>D=Dynamic S=Static K=Known Position N=No Time Recovery</td>
</tr>
<tr>
<td>2</td>
<td>Time Synchronization</td>
<td>Char</td>
<td></td>
<td>U=UTC Time G=GPS Time</td>
</tr>
<tr>
<td>3</td>
<td>Time Mark Mode</td>
<td>Char</td>
<td></td>
<td>A=Always Output Time Pulse V=Only when Valid</td>
</tr>
<tr>
<td>4</td>
<td>Maximum Time Error for which a time mark will be considered valid</td>
<td>Nsec</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>User Time Bias</td>
<td>Nsec</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Time Message Control</td>
<td>Int</td>
<td></td>
<td>0=No Message 1=830 to Control Port 2=830 to Equipment Port</td>
</tr>
<tr>
<td>7</td>
<td>Not Used</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example:**

$PMVXG,523,S,U,A,0500,000000,1,0*23

### $PMVXG,830

#### Time Recovery Results

This sentence is output approximately 1 second preceding the 1PPS output. It indicates the exact time of the next pulse, whether or not the time mark will be valid (based on operator-specified error tolerance), the time to which the pulse is synchronized, the receiver operating mode, and the time error of the last 1PPS output. The leap second flag (Field #11) is not output by older receivers.
### Field Description

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Units</th>
<th>Format</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Time Mark Valid</td>
<td>Char</td>
<td>T=Valid</td>
<td>F=Not Valid</td>
</tr>
<tr>
<td>2</td>
<td>Year</td>
<td>Int</td>
<td></td>
<td>1993-</td>
</tr>
<tr>
<td>3</td>
<td>Month</td>
<td>Int</td>
<td></td>
<td>1-12</td>
</tr>
<tr>
<td>4</td>
<td>Day</td>
<td>Nsec</td>
<td>Int</td>
<td>1-31</td>
</tr>
<tr>
<td>5</td>
<td>Time</td>
<td>HH:MM:SS</td>
<td></td>
<td>00:00:00-23:59:59</td>
</tr>
<tr>
<td>6</td>
<td>Time Synchronization</td>
<td>Char</td>
<td>U=UTC</td>
<td>G=GPS</td>
</tr>
<tr>
<td>7</td>
<td>Operating Mode</td>
<td>Char</td>
<td>D=Dynamic</td>
<td>S=Static</td>
</tr>
<tr>
<td>8</td>
<td>Oscillator Offset - estimate of oscillator frequency error</td>
<td>PPB</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Time Mark Error of last pulse</td>
<td>Nsec</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>User Time Bias</td>
<td>Nsec</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Leap Second Flag - indicates that a leap second will occur. This value is usually zero except during the week prior to a leap second occurrence, when this value will be set to +/-1. A value of +1 indicates that GPS time will be 1 second further ahead of UTC time.</td>
<td>Int</td>
<td></td>
<td>-1,0,1</td>
</tr>
</tbody>
</table>

Example:

```
$PMVXG,830,T,1998,10,12,15:30:46,U,S,000298,00003,000000,01*02
```

### 10.38 Motorola ONCORE - The Shared Memory Interface

In NMEA mode, the Oncore GPS receiver provides the user with the same information as other GPS receivers. In BINARY mode, it can provide a lot of additional information.

In particular, you can ask for satellite positions, satellite health, signal levels, the ephemeris and the almanac, and you can set many operational parameters. In the case of the VP, you can get the pseudorange corrections necessary to act as a DGPS base station, and you can see the raw satellite data messages themselves.

When using the Oncore GPS receiver with NTP, this additional information is usually not available since the receiver is only talking to the oncorder driver in NTPD. To make this information available for use in other programs, (say graphic displays of satellites positions, plots of SA, etc.), a shared memory interface (SHMEM) has been added to the refclock_oncore driver on those operating systems that support shared memory.

To make use of this information you will need an Oncore Reference Manual for the Oncore GPS receiver that you have. The Manual for the VP only exists as a paper document, the UT+/GT+/M12 manuals are available as a pdf documents at Synergy.

This interface was written by Poul-Henning Kamp (phk@FreeBSD.org), and modified by Reg Clemens (reg@dwf.com). The interface is known to work in FreeBSD, Linux, and Solaris.

#### 10.38.1 Activating the Interface

Although the Shared Memory Interface will be compiled into the Oncore driver on those systems where Shared Memory is supported, to activate this interface you must include a STATUS or SHMEM line in the `/etc/ntp.oncore` data file that looks like
STATUS < file_name >

or

SHMEM < file_name >

Thus a line like

SHMEM /var/adm/ntpstats/ONCORE

would be acceptable. This file name will be used to access the Shared Memory.

In addition, one the two keywords **Posn2D** and **Posn3D** can be added to see @@Ea records containing the 2D or 3D position of the station (see below). Thus to activate the interface, and see 3D positions, something like

SHMEM /var/adm/ntpstats/ONCORE

Posn3D

would be required.

### 10.38.2 Storage of Messages in Shared Memory

With the shared memory interface, the oncore driver (refclock_oncore) allocates space for all of the messages that it is configured to receive, and then puts each message in the appropriate slot in shared memory as it arrives from the receiver. Since there is no easy way for a client program to know when the shared memory has been updated, a sequence number is associated with each message, and is incremented when a new message arrives. With the sequence number it is easy to check through the shared memory segment for messages that have changed.

The Oncore binary messages are kept in their full length, as described in the Reference manual, that is everything from the @@ prefix thru the <checksum><CR><LF>.

The data starts at location ONE of SHMEM (NOT location ZERO).

The messages are stacked in a series of variable length structures, that look like

```c
struct message {
    u_int length;
    u_char sequence;
    u_char message[length];
}
```

if something like that were legal. That is, there are two bytes (caution, these may NOT be aligned with word boundaries, so the field needs to be treated as a pair of u_char), that contains the length of the next message. This is followed by a u_char sequence number, that is incremented whenever a new message of this type is received. This is followed by 'length' characters of the actual message.

The next structure starts immediately following the last char of the previous message (no alignment). Thus, each structure starts a distance of ‘length+3’ from the previous structure.

Following the last structure, is a u_int containing a zero length to indicate the end of the data.

The messages are recognized by reading the headers in the data itself, viz @@Ea or whatever.

There are two special cases.

1. The almanac takes a total of 34 submessages all starting with @@Cb. 35 slots are allocated in shared memory. Each @@Cb message is initially placed in the first of these locations, and then later it is moved to the appropriate location for that submessage. The submessages can be distinguished by the first two characters following the @@Cb header, and new data is received only when the almanac changes.
(2) The @@Ea message contains the calculated location of the antenna, and is received once per second. However, when in timekeeping mode, the receiver is normally put in 0D mode, with the position fixed, to get better accuracy. In 0D mode no position is calculated.

When the SHMEM option is active, and if one of Posn2D or Posn3D is specified, one @@Ea record is hijacked each 15s, and the receiver is put back in 2D/3D mode so the the current location can be determined (for position determination, or for tracking SA). The timekeeping code is careful NOT to use the time associated with this (less accurate) 2D/3D tick in its timekeeping functions.

Following the initial @@Ea message are 3 additional slots for a total of four. As with the almanac, the first gets filled each time a new record becomes available, later in the code, the message is distributed to the appropriate slot. The additional slots are for messages containing 0D, 2D and 3D positions. These messages can be distinguished by different bit patterns in the last data byte of the record.

### 10.38.3 Opening the Shared Memory File

The shared memory segment is accessed through a file name given on a SHMEM card in the /etc/ntp.oncore input file. The following code could be used to open the Shared Memory Segment:

```c
char *Buf, *file;
int size, fd;
struct stat statbuf;

file = "/var/adm/ntpstats/ONCORE"; /* the file name on my ACCESS card */
if ((fd=open(file, O_RDONLY)) < 0) {
    fprintf(stderr, "Cant open %s\n", file);
    exit(1);
}

if (stat(file, &statbuf) < 0) {
    fprintf(stderr, "Cant stat %s\n", file);
    exit(1);
}

size = statbuf.st_size;
if ((Buf=mmap(0, size, PROT_READ, MAP_SHARED, fd, (off_t) 0)) < 0) {
    fprintf(stderr, "MMAP failed\n"
    exit(1);
}
```

### 10.38.4 Accessing the data

The following code shows how to get to the individual records.

```c
void oncore_msg_Ea(), oncore_msg_As(), oncore_msg_Bb();

struct Msg {
    char c[5];
    unsigned int seq;
    void (*go_to)(uchar *);
};

struct Msg Hdr[] = { {"@@Bb", 0, &oncore_msg_Bb},
    {"@@Ea", 0, &oncore_msg_Ea},
    {"@@As", 0, &oncore_msg_As}};
```
void
read_data()
{
    int i, j, k, n, iseq, jseq;
    uchar *cp, *cp1;

    for(cp=Buf+1; (n = 256*(*cp) + *(cp+1)) != 0; cp+=(n+3)) {
        for (k=0; k < sizeof(Hdr)/sizeof(Hdr[0]); k++) {
            if (!strncmp(cp+3, Hdr[k].c, 4)) { /* am I interested? */
                iseq = *(cp+2);
                jseq = Hdr[k].seq;
                Hdr[k].seq = iseq;
                if (iseq > jseq) { /* has it changed? */
                    /* verify checksum */
                    j = 0;
                    cp1 = cp+3; /* points to start of oncore response */
                    for (i=2; i < n-3; i++)
                        j ^= cp1[i];
                    if (j == cp1[n-3]) {/* good checksum */
                        Hdr[k].go_to(cp1);
                    } else {
                        fprintf(stderr, "Bad Checksum for %s\n", Hdr[k].c);
                        break;
                    }
                }
            }
        }
        if (!strncmp(cp+3, @@Ea, 4))
            cp += 3*(n+3);
        if (!strncmp(cp+3, @@Cb, 4))
            cp += 34*(n+3);
    }
}

oncore_msg_Bb(uchar *buf)
{
    /* process Bb messages */
}

oncore_msg_Ea(uchar *buf)
{
    /* process Ea messages */
}

oncore_msg_As(uchar *buf)
{
    /* process As messages */
}

The structure Hdr contains the Identifying string for each of the messages that we want to examine, and the name of a
program to call when a new message of that type is arrives. The loop can be run every few seconds to check for new
data.

10.38.5 Examples

There are two complete examples available. The first plots satellite positions and the station position as affected by
SA, and keeps track of the mean station position, so you can run it for periods of days to get a better station position.
The second shows the effective horizon by watching satellite tracks. The examples will be found in the GNU-zipped tar file ftp://ftp.udel.edu/pub/ntp/software/OncorePlot.tar.gz.

Try the new interface, enjoy.

Reg.Clemens (reg@dwf.com), Poul-Henning Kamp (phk@FreeBSD.org)

10.39 European Automated Computer Time Services

Several European countries use the following message data format:

Data format
0000000000111111111122222222333333344444444555555555566666666666777777777
01234567890123456789012345678901234567890123456789012345678901234567890123456789
1995-01-23 20:58:51 MEZ 10402303260219950123195849740+40000500 *
A B C D EF G H IJ K L M N O P Q R S T U V W XY Z<CR><LF>
A year
B month
C day
D hour
E : normally
A for DST to ST switch first hour
B for DST to ST switch second hour if not marked in H
F minute
G second
H timezone
I day of week
J week of year
K day of year
L month for next ST/DST changes
M day
N hour
O UTC year
P UTC month
Q UTC day
R UTC hour
S UTC minute
T modified julian day (MJD)
U DUT1
V direction and month if leap second
W signal delay (assumed/measured)
X sequence number for additional text line in Y
Y additional text
Z on time marker (* - assumed delay / # measured delay)
<CR>!<LF> ! is second change!
This format is an ITU-R Recommendation (ITU-R TF583.4) and is now available from the primary timing centres of the following countries: Austria, Belgium, Germany, Italy, The Netherlands, Poland, Portugal, Romania, Spain, Sweden, Switzerland, Turkey and United Kingdom. Some examples are:

- In Germany by Physikalisch-Technische Bundesanstalt (PTB)’s timecode service. Phone number: +49 531 51 20 38
  For more detail, see http://www.ptb.de/cms/index.php?id=1786&L=1

- In the UK by National Physical Laboratory (NPL)’s Telephone Time Service. Phone number: 020 8943 6333
  For more detail, see http://www.npl.co.uk/science-technology/time-frequency/products-and-services/time/time-synchronisation-of-computers-to-utc(npl)

- In Italy by Istituto Elettrotecnico Nazionale “Galileo Ferraris” (IEN)’s CTD service. Phone number: 166 11 46 15
  For more detail, see http://www.ien.it/tf/time/Pagina42.html

- In Switzerland by Swiss Federal Office of Metrology’s timecode service. Phone number: 031 323 32 25
  For more detail, see http://www.ofmet.admin.ch/de/labors/4/Zeitvert.html

- In Sweden by SP Swedish National Testing and Research Institute ‘s timecode service. Phone number: +46 33 415783.
  For more detail, see http://www.sp.se/en/index/services/time_sync/modem_time/Sidor/default.aspx

### 10.39.1 Additional Information

Reference Clock Drivers
Symbols

–pccfreq <frequency>
  ntpd command line option, 13
–usepcc
  ntpd command line option, 13
–version
  sntp command line option, 32
–wait
  sntp command line option, 32
-4
  ntpd command line option, 11
  ntpdate command line option, 28
  ntpdc command line option, 23
  ntpq command line option, 15
-4, –ipv4
  sntp command line option, 31
-6
  ntpd command line option, 11
  ntpdate command line option, 28
  ntpdc command line option, 23
  ntpq command line option, 15
-6, –ipv6
  sntp command line option, 31
-?, –help
  sntp command line option, 31
-A
  ntpd command line option, 11
  tickadj command line option, 34
-B
  ntpdate command line option, 29
-B <bctimeout>, –bctimeout <bctimeout>
  sntp command line option, 31
-B <bdly>
  ntpdsim command line option, 40
-C <cipher>
  ntp-keygen command line option, 37
-C <snse>
  ntpdsim command line option, 40
-D <debug-level>, –set-debug-level <debug-level>
  sntp command line option, 31
-D <level>
  ntpd command line option, 12
-G
  ntp-keygen command line option, 37
-H
  ntp-keygen command line option, 37
-I
  ntp-keygen command line option, 37
-I [address | interface name]
  ntpd command line option, 12
-K <kodfile>, –kod <kodfile>
  sntp command line option, 32
-L
  ntpd command line option, 12
-M
  ntp-keygen command line option, 37
  ntpd command line option, 12
-M <steplimit>, –steplimit <steplimit>
  sntp command line option, 32
-N
  ntpd command line option, 12
-O <clk_time>
  ntpdsim command line option, 41
-P
  ntp-keygen command line option, 38
-P <priority>
  ntpd command line option, 13
-S <sim_time>
  ntpdsim command line option, 41
-S [ RSA | DSA ]
  ntp-keygen command line option, 38
-S, –step
  sntp command line option, 32
-T
  ntp-keygen command line option, 38
-T <ferr>
  ntpdsim command line option, 41
-U number, –updateinterval=number
  ntpd command line option, 13
-V <nkeys>
  ntp-keygen command line option, 38
-V <variable>
  ntpd command line option, 13
-W <fnse>
ntpsim command line option, 41
-Y <ndly>
  ntpsim command line option, 41
-Z <pdly>
  ntpsim command line option, 41
-a
  ntpd command line option, 11
-a <key>
  ntpdate command line option, 29
-a <keynum>, --authentication <keynum>
  snntp command line option, 31
-a <tickadj>
  tickadj command line option, 34
-b
  ntpd command line option, 12
  ntpdate command line option, 29
-b <bcaddress>, --broadcast <bcaddress>
  snntp command line option, 31
-b <modulus>
  ntp-keygen command line option, 37
-c
  ntpq command line option, 15
  ntptime command line option, 35
-c <command>
  ntpdc command line option, 23
-c <confiile>
  ntpd command line option, 12
-c [ RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-SHA1 | RSA-MDC2 | RSA-RIPEMD160 | DSA-SHA | DSA-SHA1 ]
  ntp-keygen command line option, 37
-c, --concurrent
  snntp command line option, 31
-d
  ntp-keygen command line option, 37
  ntpd command line option, 12
  ntpdate command line option, 29
  ntpdc command line option, 23
  ntpq command line option, 15
-d, --debug-level
  snntp command line option, 31
-e
  ntp-keygen command line option, 37
-e <authdelay>
  ntpdate command line option, 29
-e <est_error>
  ntptime command line option, 35
-f <driftfile>
  ntpd command line option, 12
-f <frequency>
  ntptime command line option, 35
-g
  ntpd command line option, 12
-g <delay>, --gap <delay>
-snntp command line option, 31
-h
  ntptime command line option, 35
-i
  ntpdc command line option, 23
  ntpq command line option, 15
-i <group>
  ntp-keygen command line option, 37
-i <jaildir>
  ntpd command line option, 12
-k <keyfile>
  ntpd command line option, 12
  ntpdate command line option, 29
-k <keyfile>, --keyfile <keyfile>
  snntp command line option, 32
-l
  ntpdc command line option, 23
-l <days>
  ntp-keygen command line option, 37
-l <logfile>
  ntpd command line option, 12
  ntptime command line option, 35
-m
  ntpd command line option, 12
-m <max_error>
  ntptime command line option, 35
-m <max_hosts>
  ntptrace command line option, 33
-m <modulus>
  ntp-keygen command line option, 37
-n
  ntpd command line option, 12
  ntpdc command line option, 23
  ntpq command line option, 15
  ntptrace command line option, 33
-n <tries>
  ntp-wait command line option, 30
-o <ntpver>, --ntpversion <ntpver>
  snntp command line option, 32
-o <offset>
  ntptime command line option, 35
-o <version>
  ntptime command line option, 35
-p
  ntpd command line option, 12
  ntpdc command line option, 23
  ntpq command line option, 15
-p <passwd>
  ntp-keygen command line option, 38
-p <pidfile>
  ntpd command line option, 12
-p <samples>
  ntpdate command line option, 29
-q

274  Index
A
addpeer
  configuration value, 26
addserver
  configuration value, 26
addtrap [ address [ port ] [ interface ]
  ntpdc command line option, 27
addvars name [ = value ] [...]
  ntpq command line option, 15
allan
  configuration value, 77
associations
  ntpq command line option, 16
authinfo
  ntpdc command line option, 27
automax
  configuration value, 71
average
  configuration value, 69
B
beacon
  configuration value, 77
broadcast
  configuration value, 26
broadcastdelay
  configuration value, 73
C
clearvars
  ntpq command line option, 15
clockbug clock_peer_address [...]
  ntpdc command line option, 26
clockstat clock_peer_address [...]
  ntpdc command line option, 25
clockvar assocID [name [ = value [...]] [...]
  ntpq command line option, 17
clrtrap [ address [ port ] [ interface]
  ntpdc command line option, 27
cohort
  configuration value, 77
cfg-from-file <filename>
  ntpq command line option, 17
configuration value
  addpeer, 26
  addserver, 26
  allan, 77
  automax, 71
  average, 69
  beacon, 77
  broadcast, 26
  broadcastdelay, 73
  ceiling, 77
  cohort, 77
  controlkey, 71
  crypto, 71
digest, 71
disable, 74
discard, 69
dispersion, 77
Index 275
NTP, Release 4.2.8p3

Index

276

maxpoll, 67
maxmem, 75
maxpoll, 67, 72
minsane, 78
mode, 67, 72
monitor, 69
mru, 75
multicastclient, 68
nonvolatile, 76
orphan, 78
orphanwait, 78
panic, 77
phone, 76
pw, 71
randfile, 71
refid, 73
requestkey, 72
reset, 76
restrict, 69
revoke, 72
rlimit, 76
saveconfigdir, 76
server, 72
setvar, 76
stacksize, 76
statistics, 80
statsdir, 80
step, 77
stepout, 77
stratum, 73
time1, 73
time2, 73
tinker, 76
tos, 77
trap, 78
trustedkey, 72
ttl, 68, 78
type, 79
version, 68
controlkey
configuration value, 71
cooked
ntpq command line option, 16
crypto
configuration value, 71
cv assocID [name [ = value [...] [...]]
ntpq command line option, 17

D
dbdebug more | less | off
ntpq command line option, 16
delay <milliseconds>
ntpdc command line option, 23
ntpq command line option, 16
delrestrict address mask [ntppor]
nptd command line option, 27
digest
configuration value, 71
disable
configuration value, 74
disable [ auth | bclient | calibrate | kernel | monitor | ntp | pps | stats]
nptd command line option, 27
discard
configuration value, 69
dispersion
configuration value, 77
dmpeers
ntptd command line option, 24
driftfile

controlkey
configuration value, 71
dbdebug more | less | off
ntpq command line option, 16
delay <milliseconds>
nptdc command line option, 23
ntpq command line option, 16
delrestrict address mask [ntppor]
nptd command line option, 27
digest
configuration value, 71
disable
configuration value, 74
disable [ auth | bclient | calibrate | kernel | monitor | ntp | pps | stats]
nptd command line option, 27
discard
configuration value, 69
dispersion
configuration value, 77
dmpeers
ntptd command line option, 24
driftfile
configuration value, 73
dscp
  configuration value, 74

E
  enable
    configuration value, 74
  enable | disable
    configuration value, 80
  enable [ auth | bclient | calibrate | kernel | monitor | ntp | pps | stats]
    ntpdc command line option, 27

F
  file
    configuration value, 79
  filegen
    configuration value, 79
  filenum
    configuration value, 76
  flag1
    configuration value, 73
  floor
    configuration value, 77
  freq
    configuration value, 77
  fudge
    configuration value, 73
  fudge peer_address [ time1 ] [ time2 ] [ stratum ] [ refid ]
    ntpdc command line option, 27

H
  help [ command_keyword ]
    ntpdc command line option, 23
  help [command_keyword]
    ntpq command line option, 15
  host
    configuration value, 71
  host <hostname>
    ntpdc command line option, 23
  host <name>
    ntpq command line option, 16
  hostnames [ yes | no ]
    ntpdc command line option, 24
  hostnames [yes | no]
    ntpq command line option, 16
  huffpuff
    configuration value, 77

I
  ident
    configuration value, 67, 71
  ifreload
  ntpdc command line option, 25
  ifstats
    ntpdc command line option, 25
  ntpq command line option, 17
  incalloc
    configuration value, 76
  includefile
    configuration value, 74
  incnmem
    configuration value, 76
  initalloc
    configuration value, 76
  initmem
    configuration value, 76
  interface
    configuration value, 74
  iostats
    ntpdc command line option, 25
    ntpq command line option, 17

K
  kerninfo
    ntpdc command line option, 25
    ntpq command line option, 17
  key
    configuration value, 67
  keyid <keyid>
    ntpdc command line option, 24
    ntpq command line option, 16
  keys
    configuration value, 72
  keysdir
    configuration value, 72
  keytype
    ntpq command line option, 16

L
  lassociations
    ntpq command line option, 17
  leapfile
    configuration value, 75
  leapsmearinterval
    configuration value, 75
  link | nolink
    configuration value, 80
  listpeers
    ntpdc command line option, 24
  logconfig
    configuration value, 75
  logfile
    configuration value, 75
  loopinfo [ oneline | multiline ]
    ntpdc command line option, 25

Index 277
M
manycastserver
  configuration value, 68
maxage
  configuration value, 75
maxclock
  configuration value, 77
maxdepth
  configuration value, 75
maxdist
  configuration value, 78
maxmem
  configuration value, 75
maxpoll
  configuration value, 67, 73
mdnstries
  configuration value, 68
memlock
  configuration value, 76
memstats
  ntpdc command line option, 25
minclock
  configuration value, 78
mindepth
  configuration value, 75
mindist
  configuration value, 78
minimum
  configuration value, 69
minpoll
  configuration value, 67, 72
minsane
  configuration value, 78
mode
  configuration value, 67, 72
monitor
  configuration value, 69
monlist [ version ]
  ntpdc command line option, 26
monstats
  ntpq command line option, 17
mreadvar assocID assocID [ variable_name [ = value[ ... ] ]
  ntpq command line option, 18
mr
  configuration value, 75
mrulist [ limited | kod | mincount=count | laddr=localaddr
  | sort=sortorder | resany=hexmask | resall=hexmask]
  ntpq command line option, 17
mrv assocID assocID [ variable_name [ = value[ ... ] ]
  ntpq command line option, 18
multicastclient
  configuration value, 68

N
nonvolatile
  configuration value, 76
ntp-keygen command line option
  -C <cipher>, 37
  -G, 37
  -H, 37
  -I, 37
  -M, 37
  -P, 38
  -S [ RSA | DSA ], 38
  -T, 38
  -V <nkeys>, 38
  -b <modulus>, 37
  -c [ RSA-MD2 | RSA-MD5 | RSA-SHA | RSA-
       SHA1 | RSA-MDC2 | RSA-RIPEMD160 | RSA-SHA | RSA-SHA1 ], 37
  -d, 37
  -e, 37
  -i <group>, 37
  -l <days>, 37
  -m <modulus>, 37
  -p <passwd>, 38
  -q <passwd>, 38
ntp-wait command line option
  -n <tries>, 30
  -s <seconds>, 30
  -V, 30
ntpdc command line option
  –pccfreq <frequency>, 13
  --usepcc, 13
  --4, 11
  --6, 11
  --A, 11
  --D <level>, 12
  --I [address | interface name], 12
  --L, 12
  --M, 12
  --N, 12
  --P <priority>, 13
  --U number, --updateinterval=number, 13
  --V <variable>, 13
  --a, 11
  --b, 12
  --c <conffile>, 12
  --d, 12
  --f <driftfile>, 12
  --g, 12
  --i <jailedir>, 12
  --k <keyfile>, 12
  --l <logfile>, 12
  --m, 12
  --n, 12
  --p <pidfile>, 12

278 Index
kerninfo, 17
keyid <keyid>, 16
keytype, 16
lassociations, 17
monstats, 17
mreadvar assocID assocID [ variable_name [ = value[ ... ]], 18
mrulist [limited | kod | mincount=count | laddr=localaddr | sort=sortorder | re-
sany=hexmask | resall=hexmask], 17
mrv assocID assocID [ variable_name [ = value[ ... ]], 18
ntversion 1 2 3 4, 16
passwd, 16
peers, 18
quit, 16
raw, 16
readvar assocID name [ = value ] [...], 18
rmvars name [...], 15
rv assocID [ name ] [...], 18
saveconfig <filename>, 18
sysinfo, 19
sysstats, 19
timeout <millseconds>, 16
writevar assocID name = value [...], 19
ntptime command line option
-c, 35
-e <est_error>, 35
-f <frequency>, 35
-h, 35
-m <max_error>, 35
-o <offset>, 35
-r, 35
-s <status>, 35
-t <time_constant>, 35
ntptrace command line option
-m <max_hosts>, 33
-n, 33
ntpversion 1 2 3 4
-ntpd command line option, 24
ntpq command line option, 16
peers
ntpd command line option, 24
ntpq command line option, 18
phone
configuration value, 76
pstats peer_address [...]
ntpd command line option, 25
pw
configuration value, 71
Q
quit
ntpd command line option, 24
ntpq command line option, 16
R
random
configuration value, 71
raw
ntpq command line option, 16
readkeys
ntpd command line option, 27
readvar assocID name [ = value ] [...]
ntpq command line option, 18
refid
configuration value, 73
requestkey
configuration value, 72
reset
configuration value, 76
ntpd command line option, 27
restrict
configuration value, 69
restrict address mask flag [ flag ]
ntpd command line option, 27
revoke
configuration value, 72
RFC
RFC 1059, 11
RFC 1119, 11, 136
RFC 1305, 6, 11, 14, 25, 33, 47, 53, 54, 71, 81, 85,
133, 136
RFC 1546, 65
RFC 1589, 124
RFC 2365, 64
RFC 2553, 47, 66
RFC 2770, 64
RFC 2783, 86, 127, 201, 205
RFC 5905, 1, 11, 30, 54, 139
RFC 5906, 46, 53, 144, 145
O
orphan
configuration value, 78
orphanwait
configuration value, 78
P
panic
configuration value, 77
passociations
ntpq command line option, 18
passwd
rlimit
  configuration value, 76
rmvars name [...]
  ntpq command line option, 15
rv assocID [ name ] [...]
  ntpq command line option, 18

S
saveconfig <filename>
  ntpq command line option, 18
saveconfigdir
  configuration value, 76
server
  configuration value, 72
setvar
  configuration value, 76
showpeer peer_address [...]  
  ntpdc command line option, 24
snmp command line option
  --version, 32
  --wait, 32
  -4, --ipv4, 31
  -6, --ipv6, 31
  -?, --help, 31
  -B <bctimeout>, --bctimeout <bctimeout>, 31
  -D <debug-level>, --set-debug-level <debug-level>, 31
  -K <kodfile>, --kod <kodfile>, 32
  -M <steplimit>, --steplimit <steplimit>, 32
  -S, --step, 32
  -a <keynum>, --authentication <keynum>, 31
  -b <bcaddress>, --broadcast <bcaddress>, 31
  -c, --concurrent, 31
  -d, --debug-level, 31
  -g <delay>, --gap <delay>, 31
  -k <keyfile>, --keyfile <keyfile>, 32
  -l <logfile>, --filelog <logfile>, 32
  -o <ntpver>, --ntpversion <ntpver>, 32
  -r, --userservedport, 32
  -s, --slew, 32
  -u <uctimeout>, --uctimeout <uctimeout>, 32
stacksize
  configuration value, 76
statistics
  configuration value, 80
statsdir
  configuration value, 80
step
  configuration value, 77
stepout
  configuration value, 77
stratum
  configuration value, 73
sysinfo
  configuration value, 68
  

T


tickadj command line option
  -A, 34
  -a <tickadj>, 34
  -q, 34
  -s, 34
  -t <tick>, 34

time1
  configuration value, 73
time2
  configuration value, 73	
timeout <milliseconds>
  ntpdc command line option, 24
  ntpq command line option, 16
timerstats
  ntpdc command line option, 25
tinker
  configuration value, 76
tos
  configuration value, 77
trap
  configuration value, 78
traps
  ntpdc command line option, 27
trustedkey
  configuration value, 72
trustedkey keyid [...]
  ntpdc command line option, 27
ttl
  configuration value, 68, 78
type
  configuration value, 79

U
unconfig peer_address [...]
  ntpdc command line option, 27
unrestrict address mask flag [ flag ]
  ntpdc command line option, 27
untrustedkey keyid [...]
  ntpdc command line option, 27

V
version
  configuration value, 68

W
writevar assocID name = value [...]

Index 281
ntpq command line option, 19