# COMSERV AND CLIENTS (Linux and Solaris version)

# INSTALLATION AND USER GUIDE

Release Version 010319
March 19 2001

Instrumental Software Technologies, Inc.

***Prepared For:***

**Kinemetrics, Inc.**
***Qunaterra, Inc.***

***Prepared By:***

***Instrumental Software Technologies, Inc.***

Offices*:*

PO Box 4554 Saratoga Springs, NY 12866
Office: (518) 580-1845 ◆ FAX: (518) 584-8875

PO Box 963 New Paltz, NY 12561
Office: (845) 256-9290 ◆ FAX: (845) 256-9299

www.isti.com

# TABLE OF CONTENT

## CONTACT INFORMATION

| CONTRACTOR INFORMATION | |
|---|---|
| Firm Name: | Instrumental Software Technologies, Inc. |
| Address: | PO Box 963<br>New Paltz, NY 12561 |
| | PO Box 4554<br>Saratoga Springs, NY 12866 |
| Telephone/Fax Numbers: | New Paltz Office: (845) 256-9290 FAX: (845) 256-9299<br>Saratoga Office: (518) 580-1845 FAX: (518) 584-8875 |
| Contact Person: | Paul Friberg (e-mail – p.friberg@isti.com) |
| Compiled by: | Dr. Ilya Dricker (e-mail - i.dricker@isti.com) |

# INTRODUCTION.

This document describes COMSERV software bundle (version comserv4linux.010319). COMSERV software bundle is a widely known tool for communication with Quanterra digitizers, receiving data from and passing the operators commands to the Quanterra digitizers. Originally it was developed for the Solaris/OS9 operating system by Quanterra, Inc. Later, Doug Neuhauser, Berkeley Seismological Laboratory, have added several useful clients and modified COMSERV core programs.

Software consists of the Communication Server **comserv**, a set of **comserv** clients that communicate with **comserv**. The bundle also includes **netmon** program, which controls the startup, shutdown and restart of **comserv** and clients.

Currently, the bundle includes the following executable modules:

- **comserv** is a Communications Server for a single Quanterra data acquisition system. The **comserv** process provides data and related information from the Quanterra unit and is used for sending commands and requests to the datalogger. There may be multiple **comserv** processes running simultaneously on the same computer (one for each station).

- **datalog** is a **comserv** client that reads data (MiniSEED records) from the **comserv** shared memory segment and writes MiniSEED data volumes to the disk files.
- **dpda** is an interactive client used to exercise all available commands from **comserv**.
- **config** goes through stations printing out station configuration information.
- **msgmon** is used as a message monitor screen for a station.
- **dataread** is a client that reads all types of data from all stations and outputs information to the terminal.
- **netmon** is a program responsible for controlling the startup, shutdown, and restart of **comserv** servers, **c2o**, **o2a** and **AlphaX86Send_client**.

This guide described a version of COMSERV bundle that has been ported to Linux operating system by Instrumental Software Technologies, Inc.

Manual pages for **comserv**, **dataread**, **dpda**, **config**, and **msgmon** are originated from Quanterra, Inc. Manual pages for **datalog** and **netmon** are originally created by Doug Neuhauser. These documents are edited by ISTI and adopted for the Linux version.

## Hardware and Operating System requirements

ISTI have tested Linux version of COMSERV bundle with the Quanterra Q730 digitiziers and the following Linux OS:

- SuSE 6.1-6.4
- RedHat 6.1
- RedHat6.2
- RedHat 7.0

ISTI have tested Solaris version of COMSERV bundle with the Quanterra Q730 digitiziers and Solaris 2.6 OS.

ISTI anticipates that the COMSERV bundle can be used with any other Quanterra digitizer, which previously have been tested with the big-endian SOLARIS version of COMSERV bundle and with virtually any flavor of Linux or Solaris operating system.

This release is "almost"[1] compatible with the version COMSERV.000518 (**comserv** version 31) of the official SOLARIS/OS9 release.

In order to use the release with Solaris, a user should use precompiled executable in the directory $(RELEASE)/bin/Solaris. To use the program with OS9 or SunOS, a user should modify the Makefile(s) and recompile the distribution.

Minimal requirements for the software bundle are:

- Intel-PC with Linux OS or a workstation with Solaris OS;
- 1 available serial port;
- A Qunaterra digitizer with available serial port;
- at least 32 Mb of memory;
- at least 10 Mb of empty disk space;

Note that **datalog** writes bulky[2] MiniSEED disk files. If you intend to routinely use **datalog**, the available disk space should be correspondingly adjusted.

[1] The only newly added feature of version 32 for is the ability of **comserv** to set the SEED byte-order bits (bit 6 and 7) in I/O and clock flags byte of the fixed MiniSEED header. In particular, **comserv** (v.32) running on big-endian computer (SUN), sets bit 7 to ON in I/O and clock flags byte of each MiniSEED header to indicate that the header byte order is big-endian. Previous versions of comserv do not modify the I/O and clock flags byte of a fixed header. See COMSERV section below for more information on setting byte-order bits.

[2] About 2-5 Mb of data/day depending on compression for a single 40 Hz channel.

## INSTALLATION

First, choose a directory where you plan to install the bundle, as in example

```
cd /home/isti
```

and copy comserv_linux.tar.gz in this direcory[3]
Next, you have to upack the archive using gunzip and tar programs

```
gunzip comserv_linux.tar.gz
tar xvf comserv_linux.tar
```

After the distribution is unpacked, the directory structure looks like in the Figure 1. Executables in the comserv/bin directory are compiled for Linux and can be used instantly. If you intend to use the software on non-Linux platform you have to compile the distribution. Refer to the Compilation section of this document.

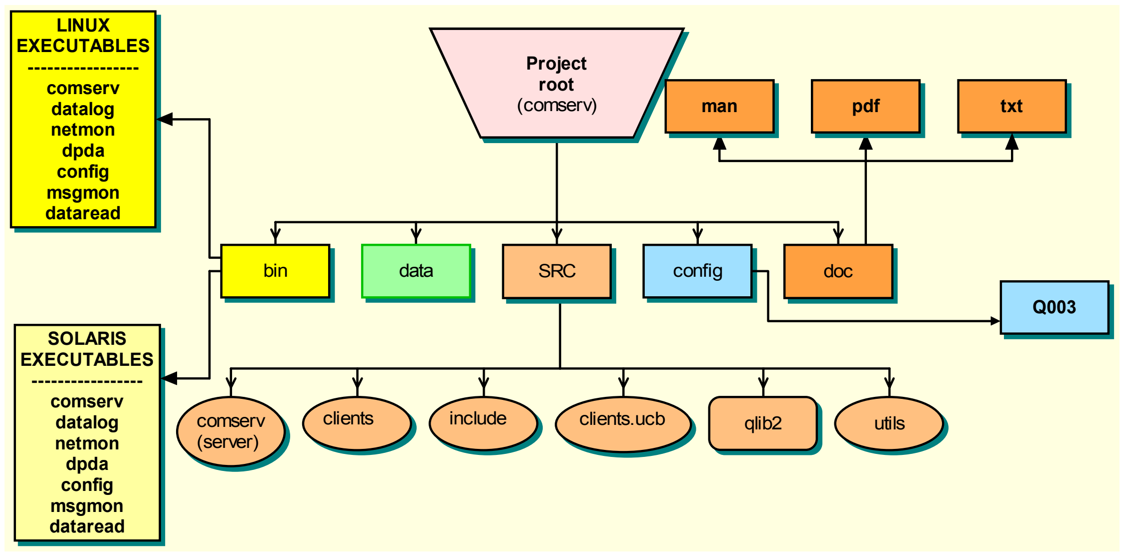


**Figure 1**

When the distribution is unpacked and (if needed) is recompiles, an operator should edit configuration files network.ini, stations.ini and station.ini[4]. Each station.ini file should reside in it's own directory. Files network.ini and stations.ini should be copied into the directory /etc (one should have super-user privileges to do this!).

[3] A good place to obtain the newest version of tar and gzip is http://rpmfind.net/linux/RPM/ .
[4] Files station.ini should be created for each instance of comserv/Qunaterra digitizer.

## CONFIGURATION

Before running the programs, a user requires to modify the configuration files, which control the modes of programs execution.
The directory comserv/config contains two configuration files.

- stations.ini
- network.ini

and an example subdirectory Q003. A subdirectory comserv/config/Q003 contains one more configuration

- station.ini.

The file station.ini from the example subdirectory is a key configuration file for a single Quanterra digitizer. It contains configuration subsections for the **comserv** and clients. The name "station.ini" is mandatory; therefore, in order to communicate with N digitizers, N comserv processes should read N different "station.ini" files. Each station.ini file must reside in it's own directory as it is shown in Figure 2.

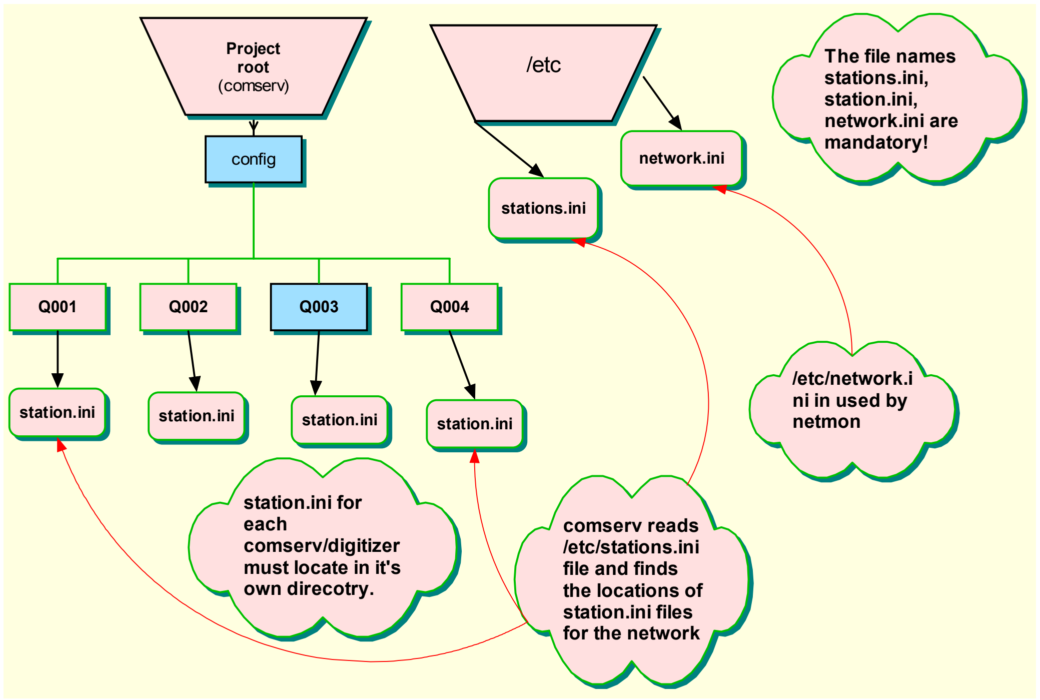

Project root (comserv)
/etc
The file names stations.ini, station.ini, network.ini are mandatory!
config
network.ini
stations.ini
Q001
Q002
Q003
Q004
/etc/network.i ni in used by netmon
station.ini
station.ini
station.ini
station.ini
station.ini for each comserv/digitizer must locate in it's own direcotry.
comserv reads /etc/stations.ini file and finds the locations of station.ini files for the network

**Figure 2**

The locations of (path to) each station.ini file are defined in stations.ini file. File stations.ini should locate in the directory /etc and should copied there by a super-user. In fact, the only purpose of the file station.ini is to contain the links to the set of station.ini files. At startup, **comserv** and client read the file /etc/station.ini, obtain the locations of station.ini files for the network and then read those files and obtain individual parameters from station.ini file.

The file network.ini contains a start-up configuration for the program **netmon** and also should be copied into the /etc directory by the super-user. If **netmon** is not going to be used, the location of the network.ini file is irrelevant.

See the following man pages on **comserv**, **netmon** and the other clients to find more information on configuration.

**Table 1**

| | Parameter file | Location | Used by programs | Description |
|---|---|---|---|---|
| 1 | network.ini | /etc/ | netmon | Scope: network. Defines the monitoring and respawning parameters of netmon daemon. |
| 2 | stations.ini | /etc/ | comserv, netmon, datalog, dataread, etc... | Scope: network. Lists stations of the network and defines the location of station-wide configuration files |
| 3 | stations.ini | $(RELEASE)/config/$(STN) | comserv, netmon, datalog, dataread, etc.... | Scope: station. Defines parameters for **comserv** and clients. |

## RUNNING PROGRAMS IN A BATCH MODE

In this mode **comserv** and clients are started and monitored by a special program – **netmon.** In a **background** daemon mode, **netmon** monitors the status of the server and blocking/reserved clients for all stations, and performs an automatic startup and restart of programs for these stations.

In automatic mode,
the command

*netmon –D –b -I STN*
where *STN* is a station name,
starts the **netmon** in daemon mode for station STN. **netmon** daemon starts and monitors the server and clients for the station STN. In a rare case when any of the clients or server crashes, the **netmon** momentarily restarts these programs. There should be no loss of data in this mode.

**netmon** is a very stable program, nevertheless, it is desirable to have tools to restart **netmon** if it is killed or crashed. More important, it is necessary to be able to automatically start netmon after hardware crash or reboot. This goal must be reached by using standard UNIX tools such as cron daemon or editing the /etc/inittab. Below we demonstrate how the task of restarting netmon is currently solved[5] at IS26.

1) A simple Born shell script **start_IS26** resides in $(RELEASE)/bin directory:

```
#!/bin/sh

lckfile=/var/lock/comserv/netmon.lock #this file is in /etc/network.ini
cmdpath=/home/sysop/cs2cd1/bin/
sleep 30
pid=`ps w -u sysop | grep netmon | grep -v grep | awk '{print $1}'`
if [ ! $pid ] ; then
            rm $lckfile
    $cmdpath/netmon -B -D -I
fi
```

This script checks for the running process **netmon** and restarts it if the process is not found. Keep in mind that due to the obvious reasons, the netmon restarting script cannot contain the word netmon in its name.

2) The crontab for the user *sysop* –owner of all comserv2CD1 processes -contains the following lines:

```
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.607 installed on Wed Oct  4 21:15:25 2000)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie
            Exp $)
```

[5] Clearly, not an optimal solution....

```
*/3 * * * * $HOME/cs2cd1/bin/start_I26 2>&1
```

Thus, the script **start_IS26** runs every three minutes doing nothing if **netmon** is alive.

3) To ensure that all lock files for all programs are removed after the hardware reboot, all lock files are collected in a single directory:

**/var/lock/comserv**

The file ***/etc/rc.d/rc.local*** contains a command

```
\rm -r /var/lock/comserv/*,
```

so all lock files are removed after every reboot of the computer.

## COMPILING THE RELEASE (OPTIONAL)

Generally, there is no need to compile the programs.
A directory bin contains two subdirectories: Solaris and Linux, which contain the precompiled executable programs for Solaris and Linux correspondingly.

Nevertheless, compilation is an easy task to perform.
In order to compile the release for Linux, cd to the $RELEASE/src directory and type

*make*

This command recursively compiles the distribution, assuming that gcc compiler and linker are installed.

To compile the release for Solaris, you have to issue the command

*make solaris.*

This command recursively *calls make –f makefile.solaris* in the src subdirectories to compile the Solaris binaries. Surely, you have to execute compilation on the Solaris platform. You must have SUN /usr/ucb/cc compiler installed on you Solaris computer. To compile the program with gcc, you have to modify the *makefile.solaris* files manually in each src directory.

The second command

*make clean*

cleans the object files. To install the newly compiled binaries in the default $(RELEASE)/bin directory, type

*make install*

Note, that make install installs the binary files in $(RELEASE)/bin directory, whereas precompiled files in $(RELEASE)/bin/Solaris and $(RELEASE)/bin/Linux directories are left untouched.

The precompiled Linux **comserv** program stores data in MiniSEED format in shared memory ring buffers according to the following convention: MiniSEED headers are stored in little-endian byte order; compressed data are stored in big-endian byte-order.

You can modify Makefile in $(RELEASE)/src/comserv directory and recompile **comserv** to force it to store MiniSEED headers in big-endian byte order. In order to do this you have to add the –D_BIG_ENDIAN_HEADER flag to the CFLAG line of $(RELEASE)/src/comserv/Makefile.

## Cleaning up obsolete MiniSEED file

If you are using **datalog** client, you will write MiniSEED records onto the disk of the computer. In case you only require to keep the most recent files (say, 7 days data archive), the way of removing the old data is to create a one-line script file and copy it into a /etc/cron.daily directory

This command should be run in cron every night at midnight.

```
find /comserv/data_directory -atime +7 -exec rm {} \;
```

All of the key characters are important.

This will remove files greater than 7 days old in the data_directory specified.

# INDIVIDUAL SOFTWARE COMPONENTS

## COMSERV

### NAME

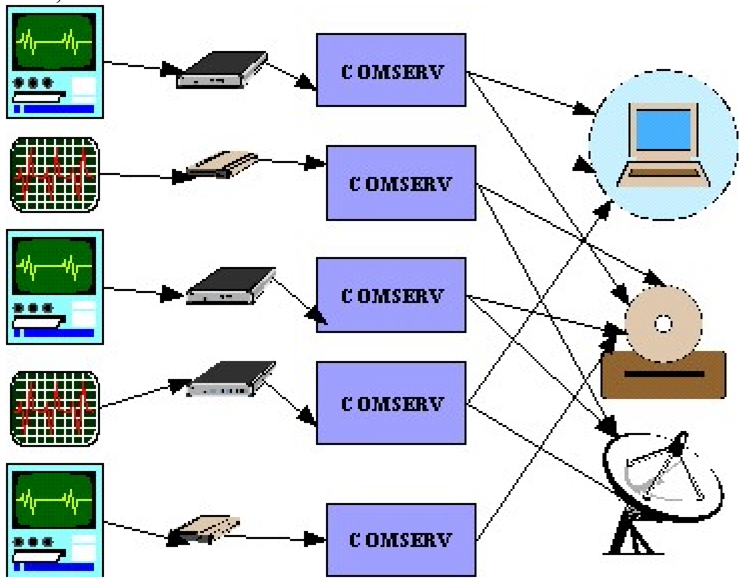
**comserv**

### SYNOPSIS

**comserv** [station name]

### DESCRIPTION

The **comserv** (Communication Server) process interfaces through a serial or Ethernet link to a single Quanterra station. There may be multiple **comserv** processes running, identified by the Station Name (4 characters maximum, padded on the right with spaces). Each client is identified by its name (4 characters maximum, padded on the right with spaces). **Comserv** provides a "crossbar switch" to allow all clients access to all stations (or any combinations of stations as the client requires). **Comserv** is a program originated by the Quanterra, Inc. and is currently supported by Quanterra User Group (QUG). In its original form **comserv** does not run on Linux computers. Refer to http://quake.geo.berkeley.edu/qug for further information. Current official **comserv** release is 31; we refer to our modification for Linux as version 32.



### IMPORTANT NOTICE:

This version of **comserv** sets the SEED header byte-order bits in the I/O and clock flags byte of the fixed header. For the description of the fixed header refer to the

Standard for the Exchange of Earthquake Data, Reference manual, p. 93, 1993. **comserv** utilizes bits 6 and 7 of the I/O and clock byte for this purpose. If the header byte order is big-endian, bit 6 is set to OFF and bit 7 is set to ON. If the header byte order is little-endian, bit 6 is set to ON; bit 7 is set to OFF. Bit setting procedure is implemented for both Linux and Solaris version of **comserv**.

## OPERATION

The **comserv** process provides data and related information from Quanterras dataloggers to the clients in SEED packets. **comserv** currently serves five types of MiniSEED packets

1. Data Packets: MiniSEED data packets consisting of:
   48-byte Fixed Data Header Blockette 1000 (data format and blocksize description) Possible additional blockettes related to the data, such as
   Blockette 1001 (enhanced timing blockette).
   Data values, in Steim1, Steim2, ... frames.

2. Detection Packets: MiniSEED data packets consisting of:
   48-byte Fixed Data Header
   Blockette 1000 (data format and blocksize description)
   Event detection blockette 2xx.

3. Calibration Packets: MiniSEED data packets consisting of:
   48-byte Fixed Data Header
   Blockette 1000 (data format and blocksize description)
   Calibration blockette 3xx.

4. Timing Packets: MiniSEED data packets consisting of:
   48-byte Fixed Data Header
   Blockette 1000 (data format and blocksize description)
   Timing packet 500.

5. LOG Packets: MiniSEED data packets consisting of:
   48-byte Fixed Data Header (data rate = 0, channel = LOG)
   Blockette 1000 (data format and blocksize description)
   ASCII console log messages.

MiniSEED data packets are stored by **comserv** in shared memory segments. Client programs read these packets from the shared memory. Clients have an access to a set of internal commands of **comserv**, which force **comserv** to serve data and interact with the Quanterra datalogger[6].

## OPTIONS

The required and the only parameter of **comserv** is [Station name]. Station name is a character string not longer than 4 characters.

[6] for more information on comserv commands, please read Quanterra document http://quake.geo.berkeley.edu/qug/soft ware/comserv/comserv.pdf .

## CONFIGURATION FILES:

### Master /etc/stations.ini file

**comserv** reads parameters from two configuration files. The master station file is **/etc/stations.ini**. Note that this name and path are embedded into **comserv**. The file stations.ini is a text file, each line starting with non-alphanumeric symbol[7] is considered to be a comment and ignored (see the example below). The master station file contains entries in the following format:

```
[NAME]
dir=station-directory
desc=station-description
source=comlink
[NAME]
etc. etc.
```

[NAME] should be replaced with real station name and must be 4 characters or less. When invoked, **comserv** reads the station name from the command line (for example, Q003) and searches for the entry [Q003] in the file /etc/stations.ini. If no such line is found, **comserv** quits. If the corresponding line is found in the /etc/stations.ini file, **comserv** reads three lines directly below. One of them must be:

- *dir=station-directory*

This is an entry point to the directory containing a file "station.ini", which contains station specific information for **comserv** and clients.

The description line is for the convenience only:

- *desc=station-description*

It is printed by **comserv** and clients to a standard output:

The line
- *source=comlink*

identifies the communication link. **comserv** does not recognize any station that does not have "source=comlink" entry in the corresponding section of the master configuration file.

Below is an example of a master configuration file for the network of two Quanterra digitizers: station Q003 and station ABC.

```
java /etc> more stations.ini
[Q003]
```

[7] except symbol '[', which identifies the beginning of a new station section of the file

```
* This is a comment since the line starts not with an alphanumeric symbol
# This is also the comment
dir=/home/comserv/config/Q003
desc=Station Q003, it's station config file is in the directory /home/comserv/config/Q003
source=comlink
[ABC]
 dir=/home/comserv/config/ABC
desc= This is the second station ABC; its parameters are in /home/comserv/config/ABC
source=comlink
```

A command

**comserv Q003**

forces **comserv** to read three lines in [Q003] section of /etc/stations.ini and then switch to reading station Q003 configuration from /home/comserv/config/Q003/station.ini.

Similarly, a command

**comserv ABC**

forces **comserv** to read the ABC configuration from /home/comserv/config/ABC/station.ini.

Keep in mind that, while the station name and path to each station configuration file are arbitrary, the file name station.ini is not. The direct consequence of this is that there is no way to have two station configuration files in the same directory.

Master /etc/stations.ini file is intended to provide a mechanism allowing several **comserv** processes invoked with different station names as arguments to find corresponding station.ini files, read station configuration and start communication with the corresponding Quanterra units. Several **comserv** processes with different station names can run simultaneously on the same computer.

## Station parameter file station.ini [comlink] section

It is generally up to a user to decide on the location of the station.ini files. Still, we believe that $RELEASE/config/[STATION_NAME] is a proper place for such files. As with /etc/stations.ini file, stations.ini is a text file; and each line starting with a non-alphanumeric symbol is considered to be a comment and ignored. A line containing [string] in the station.ini file is considered to be the beginning of the new section of the parameter file. An example is: [comlink].

**comserv** considers that the dir = [path name] line in the master /etc/stations.ini file points to the directory containing the file station.ini for the station NAME. The station.ini file must contain the [comlink] line, which indicates the beginning of the **comserv** set of parameters. Below is a list of parameters with optional lines in italics[8]:

```
[comlink]
port=path
ipport=n
updaddr=n
baud=n
parity=no | even | odd
verbosity=n
override=yes | no
notify=yes | no
flow=yes | no
station=name
seedin=yes | no
log_seed=[loc-]channel
timing_seed=[loc-]channel
segid=n
pollusecs=n
databufs=n
detbufs=n
timbufs=n
calbufs=n
msgbufs=n
blkbufs=n
reconfig=n
netto=n
netdly=n
grpsize=n
grptime=n
client1=name[,timeout]
clientn=name[,timeout]
uidnn=mask
uidnn=mask
```

- Port is the path name for the serial port

*port=path*
*EXAMPLE: port=/dev/ttyS0*
Note that port should be unique in order to use multiple Quanterra units.

- Baud is the speed (normally 9600, 19200, or 38400).

*baud=n*
*EXAMPLE: baud=38400*

- Parity is optional, and if not specified is no parity, which is Quanterra standard.

*parity=no | even | odd*
*EXAMPLE: parity=no*

- If port is not specified, then ipport must be specified and is the TCP or UDP port number to use, this number must be between 5000 and 65535.

*ipport=n*

[8] Keep in mind that some of the options are not applicable to our comserv2CD1 release. In particular, ISTI only tested comserv Linux porting in a serial, not TCP/IP mode. We also anticipate problems in comserv2orb if the default packet name "LOG" is be renamed

*EXAMPLE: ipport=17303*

- udpaddr is the IP address of the datalogger when using UDP packets, which is recommended. If for some reason you are getting excessive sequence errors TCP operation can be selected by not specifying the udpaddr parameter.

*updaddr=n*
*EXAMPLE: updaddr=123.12.24.12*

- Verbosity is 1 by default, indicating that errors and client related messages are displayed on standard output. 0 will disable the client status messages. 2 will enable one line for each packet received from the datalogger.

*verbosity=n*
*EXAMPLE: verbosity=1*

- Override is no by default, enabling station name checking from the datalogger, an error is generated if the station name in the packets does not agree with the station name you think you are talking to. If override is set to yes, then the station name in the packets will be replaced (useful for internal testing).

*override=yes | no*
*EXAMPLE: override=yes*

- Station can be used to override the station name given on the **comserv** command line, the point in doing this is left as an exercise for the reader by the original author of **comserv**.

*station=name*
*EXAMPLE: station=CHNG*

- Notify is no by default, setting to yes will enable the LINK_PKT/ULTRA_REQ handshaking required when notify is enabled on the datalogger.

*notify=yes | no*
*EXAMPLE: notify=y*

- Flow is no by default, setting to yes might enable RTS/CTS hardware handshaking on a serial port, should your system actually support it.

*flow=yes | no*
*EXAMPLE: flow=no*

- Shear and Ultra Shear dataloggers do not encode the SEED name and location for message and timing log, while Multi-Shear does. If the DA is running Multi-Shear, you can set the seedin flag to yes, **comserv** will then use the embedded names (strongly recommended for the proper work of **comserv2orb**).

*seedin=yes | no*
*EXAMPLE: seedin=yes*

- If seedin is not set to yes, log_seed can be used to override the default SEED name of "LOG" (with no location) for messages. Overwriting the default name “LOG” is not a good idea if you plan to use **comserv2orb** client. **comserv2orb** skips packets with the name “LOG” and will terminate if actual log packets will have non-“LOG” name.

*log_seed=[loc-]channel*
*EXAMPLE: log_seed=lo2*

- If seedin is not set to yes, timing_seed can be used to override the default Seed name of "ACE" (with no location) for timing blockettes. If you are using **comserv2orb,** avoid renaming “LOG” packets.

*timing_seed=[loc-]channel*
*EXAMPLE: timing_seed=ac2*

- Segid is used for the shared memory segment for that comserv. It must be unique on the system.

*segid=n*
EXAMPLE: *segid=8111*

- Pollusecs defaults to 50000 and is the number of microseconds that the **comserv** process "sleeps" between looking for new data from the serial port or for service requests from clients. This is a tradeoff between system overhead and serial port/service time.

*pollusecs=n*
*EXAMPLE: pollusecs=50000*

- Databufs is the number of 512-byte MiniSEED records that can be stored in the **comserv** and is defaulted to 20.

*databufs=n*
*EXAMPLE: databufs=20*

- Detbufs is a number of detection records that can be stored in **comserv** and is defaulted to 20.

*detbufs=n*
*EXAMPLE: detbufs=20*

- Timbufs is a number of timing records that can be stored in **comserv** and is defaulted to 20.

*timbufs=n*
*EXAMPLE: timbufs=20*

- calbufs, is a number of calibration records that can be stored in **comserv** and is defaulted to 20.

*calbufs=n*
*EXAMPLE: calbufs=20*

- msgbufs, is a number of message records that can be stored in **comserv** and is defaulted to 20.

*msgbufs=n*
*EXAMPLE: msgbufs=20*

- blkbufs is the number of block records of the appropriate type that can be stored in **comserv** and it is also defaulted to 20.

*blkbufs=n*
*EXAMPLE: blkbufs=20*

- netto is the number of seconds allowed to get any kind of valid packet from the datalogger before a TCP network connection will be closed. The default is 120 seconds and does not apply to serial or UDP connections. This option is not applicable to the current release.

*netto=n*
*EXAMPLE: netto=20*

- netdly is the delay in seconds between checking for a new TCP connection. The default is 30 seconds and does not apply to serial or UDP connections.

*netdly=n*
*EXAMPLE: netdly=20*

- reconfig is the number of sequence errors that are tolerated before a link reconfiguration is done, the default is 25.

*reconfig=n*
*EXAMPLE: reconfig=25*

- grpsize specifies how many packets from **comserv** to the datalogger need to be available to send before the whole group is sent. The default grpsize is 1, indicating that there is no grouping, packets are sent as soon as they are available.

*grpsize=n*
*EXAMPLE: grpsize=1*

- grptime is a timeout for a group, so that available packets are sent in a timely manner even if the whole group is not available, defaults to 5 seconds.

*grptime=n*
*EXAMPLE: grptime=50*

- client[n]=name1 identifies client name as a reserved client, client[n]=name2, timeout identifies client name2 as a blocking client with a timeout. Timeout is given in seconds as a parameter number two.

*clientn=name[,timeout]*
*client1=orb0, 600*
*cleint2=log, 600*
*client3=oral*
*client4=alph*

There are four types of clients: Transient, Reserved, Blocking, and Foreign. Transient clients are not identified in the configuration file and they do not occupy specific places in the client table or in service queues. Their names need not be unique (you could have multiple clients named "TEST").
Reserved clients have their names identified in the configuration file without the optional timeout value. They have a reserved slot in the client table and service queue and therefore have higher service priority than transient clients, they must also have unique names.
Like reserved clients, blocking clients are identified in the configuration file and in addition have a timeout specified in seconds. Blocking clients may actually die and come back without losing data or having data that has already been acknowledged being sent to them again. In addition, if a blocking client does not acknowledge packets it is possible that **comserv** will have to "suspend" link activity. In this instance, packets will backup in the datalogger memory and therefore it is *not useful to specify a timeout that is longer than the capability of the datalogger to buffer*. Should a blocking client not make a service call for the duration of the timeout period, **comserv** will mark that client as inactive and free up any packets that are waiting for that client. In the definition "clientn" n can be any string, the client table is filled in the order of declaration, the first declaration having the highest priority when it comes to processing service requests from clients. Blocking clients are assumed to have done an implicit "attach" call to the server when the server starts, this allows blocking clients to start after the servers and not lose data (as long as they start within the timeout value).

Foreign clients are those client processes that do not have the same User-ID as the **comserv** process. Foreign clients are normally clients being tested, either locally or from a remote user. The server cannot know when a foreign client dies, therefore the server removes a foreign client after 60 seconds of inactivity from that client. Should the same client return after it has been removed from the client table, there is only a slight processing speed penalty while the server re-installs that client. The server cannot send a wake-up signal to tell the foreign client when it has completed it's service request, therefore a foreign client will wake up on it's own every 100ms to check the completion status. Foreign clients always have access to the following internal commands of **comserv**:

*CSCM_DATA_BLK, CSCM_LINK, CSCM_CAL, CSCM_DIGII, CSCM_CHAN, CSCM_ULTRA, CSCM_LINKSTAT, CSCM_CMD_ACK,*
*and CSCM_DET_REQUEST.*

- Foreign client can have an access to the internal commands of `comserv` if there is a "uidnn=mask" entry in the [comlink] section of station.ini configuration file where "nn" is the foreign client's user ID. Mask is the sum (decimal) of the following values depending on which commands are to be enabled:

*CSCM_CLIENTS 1*
*CSCM_UNBLOCK 2*
*CSCM_RECONFIGURE 4*
*CSCM_SUSPEND and CSCM_RESUME 8*
*CSCM_TERMINATE 16*
*CSCM_SHELL 32*
*CSCM_VCO 64*
*CSCM_LINKADJ and CSCM_LINKSET 128*
*CSCM_MASS_RECENTER 256*
CSCM_CAL_START and CSCM_CAL_ABORT 512
CSCM_DET_ENABLE and CSCM_DET_CHANGE 1024
*CSCM_REC_ENABLE 2048*
*CSCM_COMM_EVENT 4096*
*CSCM_DOWNLOAD and CSCM_DOWNLOAD_ABORT 8192*
*CSCM_UPLOAD and CSCM_UPLOAD_ABORT 16384*

- *uidnn=mask*

*Example 1: uid0=32767*
*Example 2: uid503=16383*
In the first example if a user 0 (root) fires up a foreign client, `comserv` will allow this client to use any of the commands in the list above. In the second example a foreign client started by uid 503 (ilya on java.isti.com) can use any of the commands except command 16384 (CSCM_UPLOAD and CSCM_UPLOAD_ABORT)
Note that uidnn parameter is us d only for foreign clients, therefore the easiest way to avoid this parameter is to use the same user id to run both clients and server.

Below is an example of the [comlink] section of a station.ini file. The datalogger is connected via the serial port, therefore iport is not defined

```
java /etc> more /home/comserv/porting/release/config/Q003/station.ini
[comlink]
* The following are for tcp/ip da.

* NETTO: The number of seconds allowed to get any valid packet from the DA
*     before closing the connection.  Defaults is 120 seconds.
*     This value should be greater than the synctime on the dacommo to
*     avoid pointless disconnections.
* NETDLY: The number of seconds between checking for a pending client
*     connection.  Default is 30 seconds.
lockfile=/home/comserv/comserv_linux/config/lock/Q003.comserv.lock
* ipport=17303
port=/dev/ttyS0
baud=9600
parity=n
netto=120
netdly=30
* Generic parameters
grpsize=1
grptime=5
* NOTIFY works only with ultrashear release 93/05-1201 or higher.
notify=y
* parity is not really needed here.
```

```
parity=no
verbosity=2
override=yes
station=Q003
segid=8211
pollusec=50000
databufs=200
detbufs=200
timbufs=200
calbufs=20
msgbufs=200
reconfig=50
*client1=orb,1200
* turning on the dlog as a blocking client for now
*client2=dlog,3600
* uid200=32767
* uid122=32767
•    uid0=32767
•    * log_seed=lo2
* timing_seed=ac2
* This is the end of comlink section
```

## NETMON

### NAME

netmon

SYNOPSYS

```
netmon –h
netmon   [-B] [-D] [-v] [-b] [-l] [-d n] [-s | -t station_list]
    where:
        -b         Boot mode.  Assume all stations are stopped.
                   Start all stations.
        -B         Run in background.
        -D         Run in "daemon" mode in background.
        -v         Verbose mode.
        -l         Log output (stdout and stderr) to logfile in logdir.
        -s         Start all servers and blocking clients for the
                   comma-delimited list of stations.
                   'ALL' or '*' signifies all stations.
        -t         Terminate all servers and blocking clients for the
                   comma-delimited list of stations.
                   'ALL' or '*' signifies all stations.
        -d n       Set debug flag N.
        station_list
                   List of stations to start, terminate or query status.
                   List can be multiple tokens or comma-delimited list.
```

### DESCRIPTION

**netmon** is a program responsible for controlling the startup, shutdown, and restart of **comserv** servers and their dedicated blocking clients.  It is also used to manually observe the status of the servers and clients, and to explicitly issue startup and shutdown requests for the clients and server for specific stations.

**netmon** has two distinct modes of operation.  In background daemon mode, it monitors the status of the server and blocking/reserved clients for all stations, and will perform the automatic startup and restart of programs for these stations. It will also process explicit station startup and shutdown requests that were submitted to it.

### OPERATION

In interactive mode, netmon will report on the status of a station or all stations.  It is also used to submit explicit startup or shutdown request to be processed by a netmon daemon.

**Operation - Daemon mode:**

When **netmon** starts in daemon mode, it scans the network.ini config file for configuration parameters. It then scans the stations.ini file to find all stations in the network, and then scans each station's station.ini file for a [NETM] section with SERVER and CLIENT directives, and PID directives for each blocking or reserved client that **netmon** will monitor. If no [NETM] section is found or no [NETM] config directives are found, the station is ignored by the **netmon**.

If the **netmon** daemon was started with the -b (BOOT) flag, it will initially attempt to start all stations that have a run mode of *R* or *A*. Otherwise it assumes that the current station status is OK, and will only start station in response to explicit startup requests, and will restart stations that that have died since netmon's invocation.

To start (or restart) a station, **netmon** performs the following actions:

1. **netmon** starts the server program by spawning the server program with the station name as an argument.

2. After the server program has been running for SERVER_STARTUP_DELAY seconds, **netmon** queries the server for all currently registered clients. If any of the specified blocking clients are not registered, **netmon** will spawn the client programs with the station name as an argument.

Once the stations (**comserv** server and client processes) are started, it will continue to monitor all stations that are running. If a server or client process dies or is killed, **netmon** will attempt to restart the program(s).

To terminate a station, **netmon** performs the following actions:

1. It suspends the comlink for the station in order to prevent additional packets from reaching comserv by issuing a CSCM_SUSPEND command to the server.

2. It monitors the number of blocked packets for each blocking client. When the blocked packet count has reached zero, or when MAX_SHUTDOWN_WAIT seconds have expired, it terminates the client process by sending a SIGKILL signal to the client process.

3. When all blocking or reserved clients have been terminated, or when MAX_SHUTDOWN_WAIT seconds have passed since the last client was signaled, it issues a CSCM_TERMINATE command to the server to terminate the server.

## Operation - Interactive mode

**netmon** is also used to interactively query the status of a particular station or the entire network. If invoked with no arguments, or a station name, it will report on the status of the processes for the specified station, or for the entire network.

**netmon** may also be used to explicitly start or terminate the processes for a station.

1. If **netmon** is invoked with the "-s station" option, it will queue a request to the netmon daemon requesting it to start the processes for the specified station if they are not already running and if the run mode is not N (Non-runnable).

2. If **netmon** is invokes with the "-t station" option, it will queue a request to the **netmon** daemon requesting it to terminate the processes for the specified station.

The current mechanism for queuing request is to create a request file in a specified directory. This mechanism allows the standard file system permissions to be used to control that can issue **netmon** startup and terminate commands.

## Configuration files

**netmon** looks for the (new) network configuration file:
*/etc/network.ini*

### The network config file

The network config file contains the following directives, which control the operation of **netmon** (**comserv** client name NETM), comserv server, and blocking clients:

```
[netm]   logdir=/home/aq01/comserv/logs
cmddir=/home/aq01/comserv/cmds
poll_time=10
max_check_tries=5
server_startup_delay=10
client_startup_delay=10
max_shutdown_wait=30
min_notify=60
re_notify=240
notify_prog=/usr/ucb/mail -s "netmon timeout" doug
```

*logdir* is the name of the log directory where netmon places its own log, as well as those of the servers and clients that it spawns. The netmon log file will be named NETM.log. The log files for a station's process will be named STATION.server.log and STATION.client.log, where

*STATION is the upper-case station name* *server is the string "server"*
*client is the name of the client process (see below)*.cmddir *is the directory where the startup and shutdown requests are placed for* ***netmon*** *to process. The interactive commands:*

*netmon -s station(s)*
*netmon -t station(s)*

create command files in this directory to be processed by the background **netmon** process.

- *poll_time* is the time (in seconds) between the polling of each server.

- *server_startup_delay* is the maximum amount of time (in seconds) that **netmon** should allow a server process to startup. If **netmon** cannot connect to the server within this time period, **netmon** will attempt to restart the server on the assumption that the server died during initialization. This time also allow any pre-existing clients to reconnect to the server so that **netmon** can determine which clients need to be started. Set this to the *maximum* amount of time it will take to start and initialize a comserv server process on your system.

- *client_startup_delay* is the maximum amount of time (in seconds) that **netmon** should allow a client process to startup. If the client has not connected to the server within this time period, netmon will attempt to restart the client on the assumption that the client died during initialization. Set this to the *maximum* amount of time it will take to start and initialize all of the blocking client processes for a single station your system.

- *max_shutdown_wait* is the maximum time (in seconds) that netmon allows for all of the blocking clients of a single station to orderly shutdown before it signals the termination of the server process.

- *min_notify* is the default minimum timeout (in seconds) for all stations. If no successful packets have been received for *min_notify* seconds, **netmon** will consider the station "timed out", and issue a timeout notification. This value can be overridden on a per-station basis in the [netm] configuration section of a station configuration file.

- *re_notify* is the default time interval (in seconds) between successive timeout notifications for all stations. Once the first timeout notification is issued, netmon will issue subsequent timeouts every *re_notify* seconds if the station remains in a timed-out state. A value less than or equal to zero will disable renotification. This value can be overridden on a per-station basis in the [netm] configuration section of a station configuration file.

- *res_notify* is a Boolean flag indicating whether the notify program should be invoked when telemetry resumes for a station after a telemetry timeout. If the value is greater than zero, the *notify_prog* program will be run whenever telemetry resumes for a station, i.e. when a good packet has been received after a telemetry timeout for a station. A value of zero will disable telemetry resume notification. This value can be overridden on a per-station basis in the [netm] configuration section of a station configuration file.

- *notify_prog* is the default command used to issue the timeout or resume notifications. The program should expect to receive the timeout or resume message on stdin. If no value is specified for this command, **netmon** will not generate station timeout or resume notification. This value can be overridden on a per-station basis in the [netm] configuration section of a station configuration file.

**Station Config directives:**

**netmon** looks for a NETM section in the station config file station.ini for each station:

```
        [netm]
```

```
server=/home/aq01/comserv/bin/server
client1=dlog,/home/aq01/comserv/bin/datalog
inclient1==nsnf,/data/aqb1/config/bin/nsn2shear -C -c vsat.cfg
* State: A=auto-restart S=start-once R=runable N=non-runable I=ignore
state=A
server_startup_delay=10
client_startup_delay=10
max_shutdown_wait=30
min_notify=60
re_notify=240
res_notify=1
notify_prog=/usr/ucb/mail -s "netmon timeout" doug
```

It there is no [netm] section in the station config file, the station will not be monitored or controlled by netmon.server
The *server* directive specifies the pathname of the comserv server program to be run for this station. This is a required directive in the NETM section.

*stateThe* state *directive specifies the run-mode for this station.* ***netmon*** *currently supports 5 modes:*

1. *N (Non-runnable):*
   The comserv server and clients for this station may not be started up by netmon.

2. S (Start-once):
   The comserv server and clients for this station may be started by netmon upon receipt of an explicit startup command issued by a netmon request.

3. R (Runable):
   The server and blocking clients for this station should be started when the netmon daemon starts in BOOT mode, but the programs should not be restarted if they terminate.

4. A (Auto-Restart):
   The server and blocking clients for this station should be started when the netmon daemon starts in BOOT mode, and the program should be restarted if they terminate.

5. I (Ignore)
   **Netmon** should ignore this station. Do not attempt to start, stop, monitor, or report on this station.

- *client and inclient*

For each client that **netmon** should control, there should be a numbered *client* or *inclient* directive which specifies the 4-character client name and command string used to start the client program. The station name will automatically be appended to the command string as an argument. Each *client* or *inclient* directive should have a unique number, e.g. "client1", "client2", etc.

*inclient*An inclient is a program that registers as a client with comserv in order that is may be monitored and controlled by **netmon**, but in reality it performs data processing or filtering of the input data stream before it is received by **comserv**. Note that **o2a** and **AlphaX86Send_client** are not inclients. During the termination of processes for a station, these clients must be shutdown before the **comserv** link is suspended.

- *server_startup_delay*
- *client_startup_delay*
- *max_shutdown_wait*

are per-station copies of the global **netmon** directives. If these directive appear, the override the global values found in the network.ini file.

- *min_notify*

is the minimum timeout (in seconds) for this station. If no successful packets have been received for *min_notify* seconds, **netmon** will consider the station "timed out", and issue a timeout notification. If this command is omitted, or a value of 0 is specified, the network value of *min_notify* will be used for this station.

- *re_notify*

is the time interval (in seconds) between successive timeout notifications for this station. Once the first timeout notification is issued, **netmon** will issue subsequent timeouts every *re_notify* seconds if the station remains in a timed-out state. If this command is omitted, or a value of 0 is specified, the network value of *re_notify* will be used for this station.

- *res_notify*

is a Boolean flag indicating whether the notify program should be run when telemetry resumes for this station after a telemetry timeout. If the value is greater than zero, the *notify_prog* program will be run whenever telemetry resumes for this station, i.e. when a good packet has been received after a telemetry timeout for this station. A value of zero will disable telemetry resume notification. If this command is omitted, or a value of less than 0 is specified, the network value of *re_notify* will be used for this station.

- *notify_prog*

is the default command used to issue the timeout or telemetry resume notifications. The program should expect to receive the timeout or resume message on stdin. If this command is omitted, or a zero-length string is specified, the network value of *notify_prog* will be used for this station.

- *pidfile*

For each blocking or reserved client configured for a station, **netmon** will looks for a configuration section for that client within the station's station.ini file, and looks for the optional PIDFILE directive. For example, if DLOG is specified as a client to be monitored by **netmon**, **netmon** will look for the following info:

```
[dlog]
pidfile=/home/aq01/comserv/data/pid/mcc.dlog
```

The *pidfile* directive specifies the name of a file into which the client will write its process id on startup. **netmon** can use this file as a additional check to ensure that the process is running.  It is the responsibility of the client to create the specified file on startup.  If the client does not create a pidfile, the directive should not be specified.

Here are the samples of /etc/network.ini file and [netm] section of $(STATION)/station.ini file used by ISTI

```
java /home/comserv> more /etc/network.ini
[netm]
logdir=/home/comserv/comserv_linux/config/logs
cmddir=/home/comserv/comserv_linux/config/cmds
lockfile=/home/comserv/comserv_linux/config/lock/netmon.lock
server_startup_delay=10
client_startup_delay=10
max_check_tries=5
poll_time=2
min_notify=240
re_notify=21600
res_notify=1
max_shutdown_wait=30
notify_prog=/bin/mail -s "netmon notify" comserv
```

```
java config/Q003> more station.ini
*****************************************************
*     N E T M O N
*
*****************************************************
[netm]
server=/home/comserv/porting/release/bin/comserv
* datalog may have -vN option for verbose mode.
client1=dlog,/home/comserv/porting/release/bin/datalog
* State: A=auto-restart S=start-once R=runable N=non-runable I=ignore
client2=orb0,/home/comserv/porting/release/bin/comserv2orb
client3=oral,/home/comserv/porting/release/bin/orb2alpha_client
client4=alph,/home/comserv/porting/release/bin/AlphaX86Send_client
state=A
notify_prog=/bin/mail -s "netmon notify" i.dricker@isti.com
```

Sample usagebdsn> netmon

```
station=BKS2 config_mode=A target_mode=A present_mode=U
   server=comlink pid=4963 status=Good
   client=DLOG pid=4455 status=Client Dead
```

```
bdsn> netmon -v

station=BKS2 config_mode=A target_mode=A present_mode=U
   server=comlink pid=4963 status=Good
      config_dir=/data/aq01/comserv/etc/BKS2
      program=/data/aq01/comserv/bin/server
      seg=2000 nclients=1
      Ultra=1, Link Recv=1, Ultra Recv=1, Suspended=0, Data Format=QSL
      Total Packets=31928, Sync Packets=10274, Sequence Errors=6
```

```
    Checksum Errors=0, IO Errors=0, Last IO Error=0
    Blocked Packets=3, Seed Format=V2.3A, Seconds in Operation=61168
    Station Description=Berkeley Test Station, Byerly Vault
client=DLOG pid=4455 status=Client Dead
    pidfile=/data/aq01/comserv/pid/bks2.dlog
    program=/data/aq01/comserv/bin/datalog -v1
```

Background daemon:
To start the background daemon, use the command:

```
netmon -D -B -l -b &          - If no stations are running.
netmon -D -B -l &             - If restarting background netmon
                              and stations are currently running.
```

## Timeout notification

If **netmon** detects that a station has not received a valid packet in MIN_NOTIFY seconds, it considers the station to be in a timeout state, and will use the NOTIFY_PROGRAM to send a timeout notification. The timeout notification message will be send to stdin of the NOTIFY_PROGRAM. **Netmon** considers the station to remain in a timeout state until a packet has been successfully received. Once in the timeout state, **netmon** will continue to send timeout notifications ever RE_NOTIFY seconds until the station reverts to an active state, or is terminated.

## DATALOG

### NAME

**datalog**

### SYNOPSIS

datalog [-v n] [-h]
  -v n  Verbose setting
  1 = print receipt line for each packet.
  2 = display polling info.
  -h  Print brief help message for syntax.

### DESCRIPTION

The datalog client is a blocking **comserv** client that receives SEED data records from a single **comserv** server (and hence a single Quanterra station) and writes that information to disk files for archival purposes. Datalog archives the following type of information on a per-SEED channel basis:

1. DATA - SEED Data Records containing data for a specific SEED channel. The SEED Data Record may also contain blockettes defining the data format and time.

2. DETECTION - SEED Data Records containing event detection blockettes for a specific SEED channel. The Data Record will contain no data values.

3. CALIBRATION - SEED Data Records containing event calibration blockettes for a specific SEED channel. The Data Record will contain no data values.

4. OPAQUE DATA - SEEED Data Records containing Opaque Data blockettes (SEED Blockette 2000). Usually a channel that contains Opaque Data Records will not contain any other type of data. It is referred to in the configuration directives as a "blk" or blockette data.

Datalog archives the following types information on a per-station basis:

4. LOG - SEED Data Records containing all text messages from the station. The ASCII text is contained in the SEED Data Record in place of data values. The log channel is given the SEED channel name of LOG. The sample rate is 0, and the number_of_samples is used to signify the number of characters in the message

5. TIMING - SEED Data Records logs ALL clock exception blockettes.
The channel has a SEED channel name of ACE.

Since each SEED channel may have 4 distinct types of information (DATA, DETECTION, CALIBRATION, and OPAQUE DATA), we will refer to each (channel, type) pair as a "channel_type".

Each of the six types of information is assigned a extension string. Records for each channel_type are written to files located in a unique directory for that channel_type. For example, if the extension string for DATA is "D" and the extension string for DETECTION is E", the directory for BHZ DATA records would be BHZ.D and the directory for BHZ DETECTION records would be BHZ.E. All of the channel_type directories for the station are located within a single station directory.

The records for each channel_type are written to a file named "active" in the corresponding channel_type directory. The file consists of a single SEED VOL record with a blockette 8 header followed by the SEED Data records for that channel_type. The VOL record and blockette contain the SEED station, location, network and channel identifiers as well as the time of the first and last data sample (or record for non-DATA types) in the file.

## OPERATION:

Datalog reads the DLOG configuration section from the station configuration file, sets up the appropriate connection to the **comserv** server for the station as client name DLOG, and then endlessly requests SEED records from **comserv** and writes them to the appropriate disk files. On receipt of a SIGINT or SIGKILL signal, it will finish writing any outstanding records that it has received, acknowledge receipt of the data to the server, detach from the server, and terminate.

Since datalog is a blocking client, it can be stopped and started within the time limit defined for it on the CLIENT line in the [COMSERV] section with no loss of data. If the **comserv** server is stopped and started cleanly (i.e. only when it has no outstanding blocked packets for any blocking client), no packets should be lost as long as dacommo never completely uses its free buffer pool and is forced to discard packets within the Quanterra. Datalog will continue to operate correctly across a shutdown and restart of comserv.

## OPTIONS

Datalog is started with the station name as a command line argument, and an optional verbosity flag.

```
datalog [-v N] station
```

where:

-v N

specifies the level of info written to stdout.
N=0 -> normal startup/shutdown level
N=1 -> diagnostic info
N=2 -> more diagnostic info

## CONFIGURATION FILES:

### Master /etc/stations.ini file

datalog reads parameters from two configuration files. It first searches /etc/station.ini for the location of the particular station configuration file and then reads the [DLOG] and [comlink] sections of the station.ini file. Refer to the CONFIGURATION FILES section of **comserv** for more information

### Station parameter file station.ini

Configuration information for the datalog client is located in the [DLOG] section of the station configuration file.  The following sample station configuration section illustrates the configurable parameters for datalog. Detailed information on each configuration directive is shown below.

```
[DLOG]
*
* Pathnames for data directory and pid file.
*
dir=/home/aq01/data/MCC
pidfile=/home/aq01/config/etc/pid/mcc.dlog
lockfile=/home/aq01/config/etc/lock/mcc.lock
trimreclen=y
*
* Extension strings for each information type.
*
data_ext=D
detection_ext=E
calibration_ext=C
timing_ext=T
log_ext=L
blk_ext=O
*
* Channel selector for default and specific types.
* Global channel selector specified default channel selector(s) for all types.
* Specific type selector lines set the data mask (y|n) for that type,
* and optionally set specific channel selectors for that data type.
*
selector=???
data_selector=y
detection_selector=y,???
calibration_selector=y
timing_selector=y
log_selector=y
blk_selector=y
*
* Time limit specifiers for files.
* Global time limit sets default time limits for all files.
* Specific type limits set limits for either all files of that type,
* or for specific channels for that types.
*
limit=1d
data_limit=12H,HH?,HL?
data_limit=2d,V??,U??
data_limit=2d,A??,
detection_limit=1d
calibration_limit=1d
timing_limit=1d
log_limit=1d
log_blk=1d
*
save=BT?,BQ?
data_save=y,BT?,BQ?
detection_save=n
calibration_save=n
timing_save=n
log_save=n
blk_log=y
*
* Filename format template.
* %S=STATION %s=station %N=NET %n=net %C=CHAN %c=chan %X=EXTENSION
%x=extention
* %Y=year %y=yr %j=doy %m=month %d=day %H=hr %M=min
*
filename_format=%S.%N.%C.%X.%Y.%j.%H%M
```

**Datalog Configuration parameters:**

- *dir=station_dir*
  is the pathname of the station directory that will contain all of the channel_type directories.
  *EXAMPLE dir=/home/comserv/comserv_linux/data/Q003*

- *pidfile=pathname*
  is the pathname of a file used to hold the PID of the currently running Datalog program for this station. It is used by the netmon program and can be omitted if netmon is not going to be used.
  *EXAMPLE pidfile=/home/comserv/comserv_linux/config/pid/Q003.dlog*

- *lockfile=pathname*
  is a directive that specifies the pathname of a lockfile for the Datalog program for this station. If specified, **datalog** opens the file and creates an exclusive lock on the entire file while it is running. This can be used to ensure that only one copy of **datalog** is running for this station.
  *EXAMPLE: lockfile=/home/comserv/comserv_linux/config/lock/Q003.dlog.lock*

- *trimreclen=y|n*
  is an optional directive that specifies that DATA for all channels should be trimmed to the minimum record size. This option is only useful if the remote datalogger has one or more channels configured to transmit SEED records with fewer data frames than the default. For example, if a channel is defined in the datalogger with the directive COMFR=4, the data records

would contain only 4 frames of data and header, and could fit within a 256 byte SEED data record. If this option is set to Y, datalog will reset the record length of all data records for this channel to 256 bytes before archiving the data record. This can reduce wasted space in disk files. The default value is N.
*EXAMPLE: trimreclen=y*

- *data_ext=string*
  *stection_ext=string*
  *calibration_ext=string*
  *timing_ext=string*
  *log_ext=string*
  *blk_ext=string*

  define the extention strings to be used for the each type of information. The default values:

  *DATA_EXT=D*
  *DETECTION_EXT=E*
  *CALIBRATION_EXT=C*
  *TIMING_EXT=T*
  *LOG_EXT=L*
  *BLK_EXT=O (the letter "oh")*

  will be used if the directive is not specified.

- *selector=default_channel_selector*

  specifies the default selector to be used for all channel_types ifno selector is specified on a per-type basis. The default is SELECTOR=??? (or ?????).
  *EXAMPLE: selector=???*

- *data_selector=y|n[,selector_list]*
  *detection_selector=y|n[,selector_list]*
  *calibration_selector=y|n[,selector_list]*
  *timing_selector=y|n[,selector_list]*
  *log_selector=y|n[,selector_list]*
  *blk_selector=Y|N|,selector_list]*

  specifies whether the type of information should be acquired from the server (i.e. the value of the channel mask), and optionally specifies a comma-delimited selector list for that type. If no directive is given, the default is to acquire that type information using the default_channel_selector specified by the SELECTOR directive. If a value of N is specified for the channel mask, the selector list is ignored, since no packets of that type will be received from the server.

This allows you to selectively configure what channels and types of information you want to acquire. For example, you could chose to acquire only BH? and LH? data, but acquire event detections from all channels.

NOTE: If you configure datalog as a BLOCKING CLIENT in the [comlink] section of the station configuration file, you should NOT use ANY Of the SELECTOR directives to limit the type or channels of information that you receive from the server. If you specify ANY type of info or channels that you do not want to receive from the server, the server will maintain those undelivered packets in the server buffer, and the server may eventually block (due to a lack of free packets in the server) and may stop receiving packets from the datalogger. If you configure

datalog as a blocking client, use the SAVE directives to select the type of info and channels that you want to write to disk.
*EXAMPLE: data_selector=y*
*detection_selector=y,???*
*calibration_selector=y*
*timing_selector=y*
*log_selector=y*

- *save=default_channel_selector*

  specifies the default selector to be used for selecting data to be written to disk for all channel_types if no selector is specified on a per-type basis for writing. The default is SAVE=??? (or ?????).
*save=???*

- *data_save=y|n [,selector_list]*
  *detection_save=y|n [,selector_list]*
  *calibration_save=y|n [,selector_list]*
  *timing_save=y|n [,selector_list]*
  *log_save=y|n[,selector_list]*
  *blk_save=y|n[,selector_list]*

  specifies whether the type of information should be written to disk, (i.e. a channel mask for writing to disk), and optionally specifies a comma-delimited selector list for that type. If no directive is given, or a channel_mask of "Y" is given with no selector list, all packets of that type that are received from the server and match the SAVE selector mask will be written to disk. If a value of N is specified for the channel mask, the selector list is ignored, and no packets of that type will be written to disk.

  This allows you to selectively configure what channels and types of information you want to write to disk. For example, you could chose to write only BH? and LH? data, but write event detections from all channels.
*EXAMPLE: data_save=y*
*detection_save=y*
*calibration_save=y*
*timing_save=y*
*log_save=y*

- *limit=time_span*

  specifies the global default time span for all active files. Records are logged to an active file (named "active") in each channel_type directive. When a SEED record for a channel_type is received, and that record exceeds the time span of the active file, the active file is closed and renamed, a new active file is started, and the SEED record is logged to the new active file.
 A time_span can be specified as either <n>d (for <n> days) or <n>H (for <n> Hours). Currently, the Hours limit must be <= 24 and must integrally divide 24 with no remainder in order that one of the file boundaries will be approximate a day boundary. The default LIMIT is 1d.
*EXAMPLE: limit=1d*

- *data_limit=time_span[,selector_list]*
  *detection_limit=time_span[,selector_list]*
  *calibration_limit=time_span[,selector_list]*
  *TIMING_LIMIT=time_span[,selector_list]*
  *Log_limit=time_span[,selector_list]*
  *Blk_limit=time_span[,selector_list]*

specifies the default time span either for all channels of the specified type, or for all channels that match the selector list for the specified type. For example, DATA_LIMIT=12H specifies that all DATA files should span 12 hours. DATA_LIMIT=12H,HH?,HL? DATA_LIMIT=2d,V??,U?? specifies that DATA files for HH? and HL? channel should span 12 hours, and DATA files for V?? and U?? channels should span 2 days. There may be multiple directives for each information type.

If a limit for a channel is not explicitly specified for a given type, the limit for that channel is the default limit for that type. If a channel matches a channel_selector on more that one LIMIT of the same type , the last LIMIT directive that matches that channel will be used. If no default limit is specified for a type, the default limit for the type is the global limit.

*EXAMPLE: data_limit=1H,HH?,HL?*
*data_limit=1H,V??,U??*
*data_limit=1H,A??,*
*detection_limit=1d*
*calibration_limit=1d*
*timing_limit=1H*
*log_limit=1H*

- *filename_format=template*

specifies the format template used to construct the filename for inactive files. When the time_span is exceeded for an active file, a new filename is constructed using the template, and the file is renamed to the new filename. The template can contain the following special conversion characters:

*%S = STATION NAME (upper case)*
*%s = station name (lower case)*
*%N = NETWORK NAME (upper case)*
*%n = network name (lower case)*
*%C = CHANNEL NAME (upper case)*
*%C = channel name (lower case)*
*%L = LOCATION NAME (upper case)*
*%l = location name (lower case)*
*%X = TYPE EXTENSION (upper case)*
*%X = type extension (lower case)*
*%Y = 4 digit year*
*%y = 2 digit year*
*%j = 3 digit day-of-year*
*%m = 2 digit month*
*%d = 2 digit day-of-month*
*%H = 2 digit hour*
*%M = 2 digit minute*

Any other characters in the template are used verbatim in the filename. The date and time information refer to the time of the first record within the file, NOT to the beginning time of the time_span. The default filename template is:

*%S.%N.%C.%X.%Y.%j.%H%M*

If you use the location name (%L or %l) in the filename template, be SURE that the location string is not blank (which is the default SEED location). Otherwise, you will have blanks in your filename.

Examples:

The data logging directory for station MCC may contain the following directories, one for each channel.type of info:

*LOG.L BHZ.D BHN.DBHE.DLHZ.D LHN.DLHE.D VHZ.D VHN.DVHE.D*
*ACE.T BHZ.E LHZ.E VHZ.E*

Within each directory, there will be an "active" file as well as older non-active files. for example, the BHZ.D directory may contain the following files:

*active MCC.BK.BHZ.D.1994.304.0000 MCC.BK.BHZ.D.1994.305.0000*

Finally, here is an example of datalog section from our test:

```
[dlog]
* Pathnames for data directory, program, and pid file.
dir=/home/comserv/comserv_linux/data/Q003
pidfile=/home/comserv/comserv_linux/config/pid/Q003.dlog
lockfile=/home/comserv/comserv_linux/config/lock/Q003.dlog.lock
* Selector specifies general selectors for data, detections, and calibrations.
* Specific type selector lines set the data mask (y|n) for that type,
* and optionally set specific selectors for that data type.
selector=???
data_selector=y
detection_selector=y,???
calibration_selector=y
timing_selector=y
log_selector=y
*
* ATWC Q730s are shipping 256 byte packets, trim them on disk
*
trimreclen=y
*
save=???
data_save=y
detection_save=y
calibration_save=y
timing_save=y
log_save=y
*
limit=1d
data_limit=1H,HH?,HL?
data_limit=1H,V??,U??
data_limit=1H,A??,
detection_limit=1d
calibration_limit=1d
timing_limit=1H
log_limit=1H
*
data_ext=D
detection_ext=E
calibration_ext=C
timing_ext=T
```

```
log_ext=L
* Format specifier for filenames:
* %S=STATION %s=station %N=NET %n=net %C=CHAN %c=chan %X=EXTENSION
%x=extention
* %Y=year %y=yr %j=doy %m=month %d=day %H=hr %M=min
filename_format=%S.%N.%C.%X.%Y.%j.%H%M
```

## DPDA

### NAME

**Dpda**

### SYNOPSIS

dpda STN
Where STN is a required station code (up to 4 characters long)

### DESCRIPTION

This is an interactive client used to exercise all available commands available from **comserv** (excluding reading data records and blockettes). This client shows how to set up the data structures and execute the commands. Its client name is "CMDS" in the client table of comserv, but there is no need to enter any **dpda** parameters into the $(RELEASE)/config/$(STATION)/station.ini file, since **dpda** is a transient **comserv** client

### OPERATION:

The **dpda** client process reads master configuration file, sets up the appropriate connection to the **comserv** server for the station as client name CMDS, and prints its menu:

```
comserv@java:~/porting/release/bin > dpda Q003
Edition 20 Station Q003--------------------------------------------------------------------------------
0=Exit                    1=Link Status            2=Link Format
 3=Digitizer Info         4=Ultra Info             5=Calibrator Info
 6=Channel Info           7=Detector Info          8=Client Info
 9=Unblock packets        10=Reconfigure Link      11=Suspend Link
12=Resume Link            13=Command Clear         14=Command Status
15=Terminate Server       16=Shell Command         17=Set VCO Value
18=Change Link Settings   19=Mass Recenter         20=Calibrate
21=Detector Enable        22=Detector Change       23=Recording Enables
24=Comm Event             25=Xfer File to Host     26=Xfer File to DA
27=Server Link Settings   28=Flooding Control
Command :
```

A user communicates with dpda/comserv/datalogger via menu entries. Detailed description of each out of 28 menu entries is out of scope of the current release of documentation. Most of them are rather intuitive. For the additional details please refer directly to Quanterra, Inc. (http://www.kmi.com ). The mechanism of those requests is generally straightforward. For example, we ask dpda to provide us with an information on Link Status. To do this we type:

```
Command : 1[ENTER]
```

This causes dpda to issue and send to **comserv** a command `CSCM_LINKSTAT`[9]. **Comserv** checks if the information about link status is available and returns the link-related data structure via command buffer (shared memory):

```
-----------------------------------------------------------------------
 Ultra=1, Link Recv=1, Ultra Recv=1, Suspended=0, Data Format=QSL
 Total Packets=211823, Sync Packets=0, Sequence Errors=2834
 Checksum Errors=371, IO Errors=0, Last IO Error=0
 Blocked Packets=0, Seed Format=V2.3B, Seconds in Operation=268739
 Last Good Packet Received at=01:11:43.141783 Tue Feb 29 2000
 Last Bad Packet Received at=01:04:31.121795 Tue Feb 29 2000
 Station Description=Test of comserv2alpha release  at ISTI North (Linux porting
 Polling=50000us, Reconfig=50, Net Timeout=120, Net Polling=30
 Group Size=1, Group Timeout=5
```

Such requests as 1 (Link Status) or 2 (Link Format) do not require **comserv** to interact with the datalogger software, while the others (for example, 19 (Mass Recenter) or 20 (Calibrate)) require dpda to contact the datalogger software and even alter the hardware status.

## CONFIGURATION FILES:

### Master /etc/stations.ini file

Dpda requires a master configuration file /etc/stations.ini. No particular modifications of station.ini file are required for the proper work of dpda

[9] for more information on comserv/datalogger commands, please read Quanterra document http://quake.geo.berkeley.edu/qug/soft ware/comserv/comserv.pdf .

## MSGMON

### NAME

**msgmon**

### SYNOPSIS

msgmon STN
Where STN is a required station code (up to 4 characters long)

### DESCRIPTION

This program can be used as a message monitor screen for a station, for instance, when you are using the **dpda** program to send shell commands. It shows only messages from the specified station. Its client name is "MSGM".

### OPERATION:

The **msgmon** client process reads master configuration file, sets up the appropriate connection to the **comserv** server for the station as client name MSGM, and prints user's menu:

```
 Startup option (0 = First, 1 = Last, 2 = At Time):
```

0 means that **msgmon** prints **comserv** messages starting with the first (in time) available. 1 indicates that **msgmon** will wait for the next available message after it is attached to **comserv**. Finally, 2 indicates that user would like to check messages which were generated within the particular time range (in seconds).

Here is the example
msgmon Q003

```
Startup option (0 = First, 1 = Last, 2 = At Time) : 2
Number of seconds previous to current time to start : 1000
FROM AQSAMPLE: A0[10]/02/29 04:14:05  UM1:     31.382      0.001      0.001
FROM AQSAMPLE: A0/02/29 04:14:05  UM2:     31.382      0.001      0.001
FROM AQSAMPLE: A0/02/29 04:14:05  UM3:     32.395      0.001      0.001
```

Here is the other example:

```
comserv@java:~/porting/release/bin >
comserv@java:~/porting/release/bin > msgmon Q003
Startup option (0 = First, 1 = Last, 2 = At Time) : 0
FROM CLOCK: DATE=2000/02/28 TIME=15:28:00 STATUS=3D Fix
```

[10] Note that the year is presented as A0 instead of 00. This is a Y2K Quanterra digitizer glitch. This message is generated by the aqsample, internal datalogger program. Then this message is passed to **comserv**/**msgmon** as a character string without any changes. The Y2K glitch seems to be of no harm and Quanterra has indicated to us that they are aware of this problem

```
FROM CLOCK: LAT=433.270N LONG=7343.722W HEIGHT=94M DOP=4.5
FROM CLOCK: DOPTYPE=2D Ant OK VISIBLE=12 TRACKING=5 STATUS=2D Fix
FROM CLOCK: SAT=1 0 $02  SAT=18 0 $02  SAT=19 40 $0A  SAT=27 39 $0A
FROM CLOCK: SAT=8 42 $0A  SAT=3 37 $0A  SAT=13 45 $0A  SAT=31 0 $02
FROM CLOCK: GPS3 Driver Version number 3
FROM CLOCK: DATE=2000/02/28 TIME=15:36:00 STATUS=2D Fix
FROM CLOCK: Quality=0
FROM CLOCK: Quality=3
FROM CLOCK: LAT=433.256N LONG=7343.729W HEIGHT=94M DOP=6.7
FROM CLOCK: DOPTYPE=3D Ant OK VISIBLE=12 TRACKING=5 STATUS=3D Fix
FROM CLOCK: SAT=8 45 $0A  SAT=18 0 $02  SAT=19 40 $0A  SAT=27 46 $0A
FROM CLOCK: SAT=15 35 $0A  SAT=3 0 $02  SAT=13 45 $0A  SAT=31 0 $02
FROM CLOCK: GPS3 Driver Version number 3
FROM CLOCK: DATE=2000/02/28 TIME=15:53:00 STATUS=3D Fix
FROM CLOCK: LAT=433.202N LONG=7343.803W HEIGHT=104M DOP=6.4
FROM CLOCK: DOPTYPE=2D Ant OK VISIBLE=12 TRACKING=4 STATUS=2D Fix
FROM CLOCK: SAT=8 47 $0A  SAT=18 0 $02  SAT=19 39 $0A  SAT=27 44 $0A
FROM CLOCK: SAT=15 0 $02  SAT=3 0 $02  SAT=13 43 $0A  SAT=31 0 $02
FROM CLOCK: GPS3 Driver Version number 3
FROM CLOCK: DATE=2000[11]/02/28 TIME=15:55:00 STATUS=2D Fix
```

## CONFIGURATION FILES:

### Master /etc/stations.ini file

msgmon requires a master configuration file /etc/stations.ini. No particular modifications of station.ini file are required for the proper operation of this program.

[11] Note that GPS clock generates the correct year (2000)….

## CONFIG (CLIENT)

### NAME

**config**

### SYNOPSIS

config STN
Where STN is a required station code (up to 4 characters long)

### DESCRIPTION

This program goes through all stations (or the specified station) showing configuration information available at that time about the stations. It is useful if you want to save a "reference configuration" for a station on hard copy or disk for later reference. Its client name is "CONF".

### OPERATION:

The **config** client process reads master configuration file, sets up the appropriate connection to the **comserv** server for the station as client name CONF, and outputs  a lot of configuration information:

```
comserv@java:~/porting/release/bin > config Q003 | more
[Q003] Status=Good
CSIM_LINKSTAT for [Q003]
  Ultra=1, Link Recv=1, Ultra Recv=1, Suspended=0, Data Format=QSL
  Total Packets=221296, Sync Packets=0, Sequence Errors=2848
  Checksum Errors=372, IO Errors=0, Last IO Error=0
  Blocked Packets=0, Seed Format=V2.3B, Seconds in Operation=281293
  Station Description=Test of comserv2alpha release  at ISTI North (Linux porting
  Polling=50000us, Reconfig=50, Net Timeout=120, Net Polling=30
  Group Size=1, Group Timeout=5
CSIM_LINK for [Q003]
  Window Size=16, Total priority levels=33, Message Priority=16
  Detection Priority=16, Timing Priority=16, Cal Priority=16
  Resend Time=10  Sync Time=10  Resend Pkts=1
  Net Restart Dly=120  Net Conn. Time=60  Net Packet Limit=-1
  Group Pkt Size=1  Group Timeout=0
CSIM_DIGI for [Q003]
  Digitizer=Q4120, Version=35/07-0601
  Clock Message=Message to send to GPS? ([E]0=off, 1=on, 2=status, 3=cold start)
…………………………………..
etc…….
```

**config** is somewhat a non-interactive version of **dpda** with a limited functionality (only commands which  display information are send by **config** to **comserv** , while **dpda** allows to interact and modify settings on the datalogger).

### CONFIGURATION FILES:

### Master /etc/stations.ini file

**config** requires a master configuration file /etc/stations.ini. No particular modifications of station.ini file are required for the proper operation of this program.

## DATAREAD

### NAME

**dataread**

### SYNOPSIS

dataread   -v [n] STN
where STN is a required station code (up to 4 characters long)
-v n    Verbose setting
1 = print receipt line for each packet.
2 = display polling info.

### DESCRIPTION

This is an example of a client that reads all types of data from all stations (or only the indicated station if specified on the command line). It doesn't do anything with the data other than show the Channel ID and time of reception on the screen unless the -v option is used, in which case additional information is shown. Its client name is "DATA".

### OPERATION:

The **dataread** client process reads master configuration file, sets up the appropriate connection to the **comserv** server for the station as client name DATA and prints out the statistics of LOG and DATA packets, which are currently in the comserv. –v flag defines the level of the details.

```
dataread Q003
[Q003] < 5> Channel=EHZ Seq=059919 Received at=04:51:33.571840 Tue Feb 29 2000
[Q003] < 6> Channel=HHZ Seq=059920 Received at=04:51:36.191874 Tue Feb 29 2000
[Q003] < 7> Channel=HHN Seq=059920 Received at=04:51:36.731854 Tue Feb 29 2000
[Q003] < 8> Channel=HHE Seq=059920 Received at=04:51:37.271843 Tue Feb 29 2000
[Q003] < 9> Channel=EHZ Seq=059920 Received at=04:51:37.811848 Tue Feb 29 2000
```

```
dataread –v 2 Q003
[Q003] < 0> Channel=EHZ Seq=059940 Received at=04:53:41.141848 Tue Feb 29 2000
    Time=04:49:36.323067 Tue Feb 29 2000   Blks=2   Samples=616
    Activity Flags=--------   IO Flags=--------   Data Quality Flags=Q-------
    Encoding Format=Steim2   Frequency=100.00Hz  Clock Quality=17%
[Q003] < 0> Channel=HHZ Seq=059941 Received at=04:53:45.341850 Tue Feb 29 2000
    Time=04:49:42.483067 Tue Feb 29 2000   Blks=2   Samples=616
    Activity Flags=--------   IO Flags=--------   Data Quality Flags=Q-------
    Encoding Format=Steim2   Frequency=100.00Hz  Clock Quality=17%
[Q003] < 1> Channel=HHN Seq=059941 Received at=04:53:45.821856 Tue Feb 29 2000
    Time=04:49:42.483067 Tue Feb 29 2000   Blks=2   Samples=616
    Activity Flags=--------   IO Flags=--------   Data Quality Flags=Q-------
    Encoding Format=Steim2   Frequency=100.00Hz  Clock Quality=17%
[Q003] < 0> Channel=HHE Seq=059941 Received at=04:53:46.361840 Tue Feb 29 2000
    Time=04:49:42.483067 Tue Feb 29 2000   Blks=2   Samples=616
    Activity Flags=--------   IO Flags=--------   Data Quality Flags=Q-------
    Encoding Format=Steim2   Frequency=100.00Hz  Clock Quality=17%
```

## CONFIGURATION FILES:

### Master /etc/stations.ini file

**config** requires a master configuration file /etc/stations.ini. No particular modifications of station.ini file are required for the proper operation of this program.

## ACKNOWLEDGMENT

ISTI would like to thank Dr. Joseph Steim (Quanterra, Inc.) and Doug Neuhauser (UCB) for help and advice with the COMSERV Linux porting. We acknowledge an original support of the porting efforts by Manfred Hanger and Gernot Hartmann (BGR). Mr. Kim (Korea) provided us with the original testing of **comserv** UDP and TCP modes with Linux. Slava Bereza (CTBTO) pointed to a bug in Linux version of comlink.c.