NAQSServer
Version 2.1

User Guide

Nanometrics
SEISMOLOGICAL INSTRUMENTS

Kanata, Ontario • Canada

NAQSServer Version 2.1 User Guide

Nanometrics Inc.
250 Herzberg Road
Kanata, Ontario, Canada K2K 2A1
Tel +1 613-592-6776
Fax +1 613-592-5929
Email info@nanometrics.ca
www.nanometrics.ca

Part number 14826R4
Release date 2008-03-28

About This User Guide

Document Scope

This user guide provides instructions for installing, configuring, and running NAQSServer.

- Chapter 1 Getting Started – Installation instructions and an overview of the components and functionality of NAQSServer
- Chapter 2 Configuring NAQSServer – Instructions on how to configure NAQSServer using the `Naqs.ini` and `Naqs.stn` files
- Chapter 3 Running NAQSServer – Information on stopping and starting NAQSServer, using the run-time commands, and monitoring the operation of NAQSServer
- Appendix A Configuration File Examples – Examples of `Naqs.ini` and `Naqs.stn` files

Document Conventions

Essential and Supplementary Information:		
⚠	**Warning**	A Warning is essential information that explains (1) a risk of injury or a risk of irreversible damage to data, an operating system, or equipment; and (2) preventive action.
⚠	**Caution**	A Caution is essential information that explains (1) a risk of damage to equipment, data, or software where the recovery is likely to be troublesome; and (2) preventive action.
✎	**Note**	A Note is an explanation or comment that is related to the main text but is not essential information.
Links:		
	blue text	An external link; for example http://www.nanometrics.ca A link to information within the document.
Text Conventions:		
	bold text	Buttons on the graphical user interface (GUI).
	italic text	Variables such as parameter names and value placeholders
	`courier text`	File names and paths; for example ... `/nmx/user/trident.rsp`
	`courier bold text`	Input commands shown exactly as they must be entered at the prompt For example: ... and then type mkdir **`$APOLLO_LOCATION/config`**.

Contents

Tables

Chapter 1
Getting Started

NAQSServer is a data acquisition system designed to receive, process, and store serial data, seismic data, and state-of-health information, and process calibration commands. NAQSServer can be used with networks ranging in size from a few instruments to those comprising 100 or more remote instruments.

1.1 About NAQSServer

NAQSServer is designed for use with Nanometrics data acquisition and communications instruments. Supported instruments include the following:

- HRD – (24-bit high resolution digitiser with onboard GPS)
- RM3 – (serial repeater/multiplexer)
- RM4 – (serial-to-IP bridge/multiplexer)
- Carina – (Libra VSAT hub)
- Carina105 (Libra VSAT hub)
- Lynx – (Libra remote VSAT station with integrated digitiser)
- Cygnus – (Libra remote VSAT station with integrated TimeServer, NMXbus, and serial data ports)
- Cygnus205 (Libra remote VSAT station with NMXbus and serial data ports)
- Europa – (HRD digitiser with integrated radio or Ethernet)
- Janus – (Callisto remote communications controller with integrated TimeServer, NMXbus, and serial data ports)
- Trident – (24-bit high resolution digitiser with NMXbus)
- Trident305 (24-bit high resolution digitiser with NMXbus)
- Taurus (portable seismic data logger with 24-bit high resolution digitiser)

Note Starting with NAQSServer version 2.1, the use of NpToNmxp version 3.1 or later is required to receive data from any of the following new generation devices: Carina105, Cygnus205, Trident305, and Taurus.

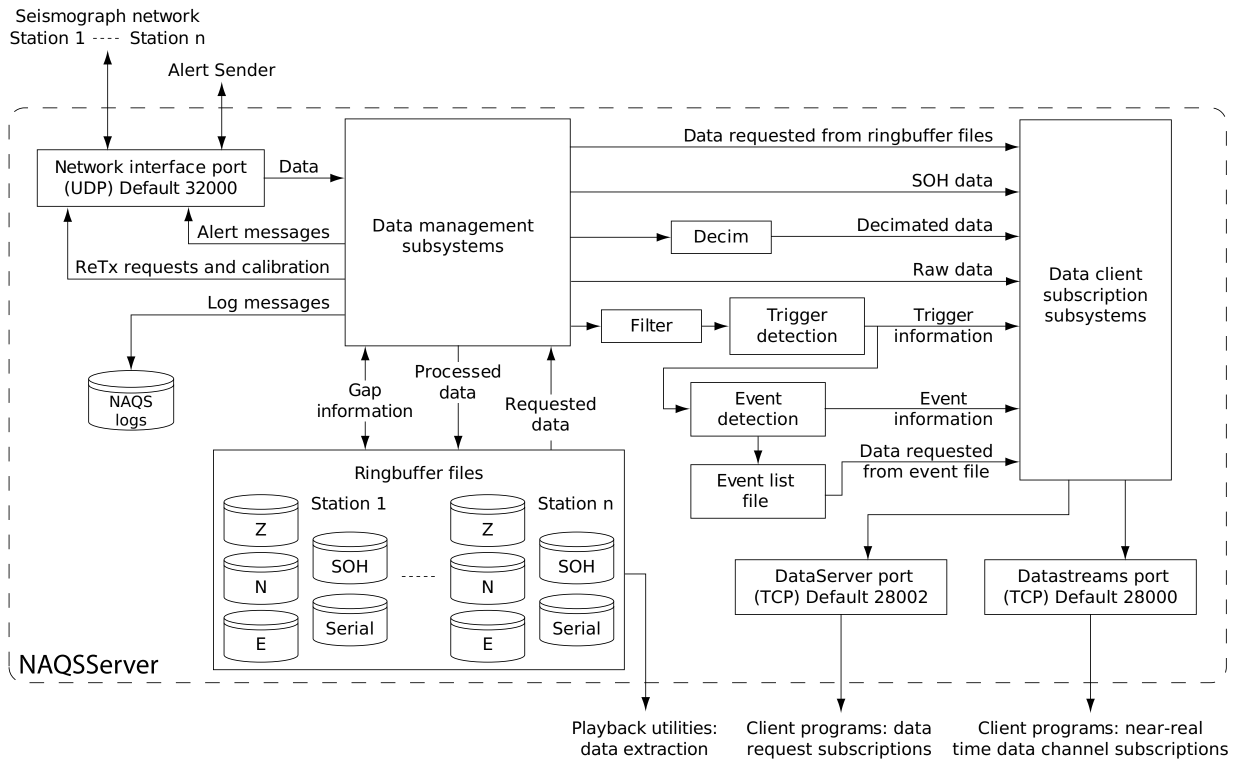
1.1.1 NAQSServer Data Flow

NAQSServer uses a channel-oriented telemetry design; each data channel (or seismic component) is transmitted and archived as a separate data stream. Data for each channel (component) are tagged, timestamped, and compressed by the digitiser and are sent to NAQSServer as fixed-length data packets. Because the data packets are tagged by the originating instrument, the data routing is unimportant; NAQSServer uses the source tag to identify each packet and direct it to the correct channel archive. The use of TCP/IP (UDP) for packet transport results in a very modular system design, and minimizes hardware requirements and system dependencies at the central acquisition site.

The primary purpose of NAQSServer is to collect and store incoming data in ringbuffers, from which it can be extracted and post-processed by other programs such as Nanometrics Atlas seismic data analysis package. NAQSServer ensures maximum data recovery by automatically requesting retransmission of any packets which are lost or corrupted during transmission. NAQSServer also performs trigger and event detection, and provides online access to serial, seismic, and state-of-health data via TCP/IP subscription.

Figure 1-1 shows an overview of the NAQSServer data flow.

Figure 1-1 NAQSServer data flow



Seismograph network
Station 1 ---- Station n
Alert Sender
Network interface port (UDP) Default 32000
Data
Alert messages
ReTx requests and calibration
Log messages
NAQS logs
Data management subsystems
Gap information
Processed data
Requested data
Ringbuffer files
Station 1
Station n
Z
N
E
SOH
Serial
Data requested from ringbuffer files
SOH data
Decim
Decimated data
Raw data
Filter
Trigger detection
Trigger information
Event detection
Event information
Event list file
Data requested from event file
Data client subscription subsystems
DataServer port (TCP) Default 28002
Datastreams port (TCP) Default 28000
NAQSServer
Playback utilities: data extraction
Client programs: data request subscriptions
Client programs: near-real time data channel subscriptions

1.1.2 Online Data Access

NAQSServer provides near real-time online access to serial data, seismic time-series, triggers, event messages, and state-of-health data via TCP/IP subscription. For more information, see the chapter on Private Data Streams in the Nanometrics Data Formats user guide.

DataServer provides access to historical data stored in ringbuffers for post-processing and, thus, complements the real-time data service provided by NaqsServer online data streams. For more information, see the DataServer user guide.

1.1.3 Alert Messages

NAQSServer issues messages alerting the network manager or other interested parties of the acquisition system status and significant events. See Section 3.3.2 “Interpreting NAQSServer Alert Messages” on page 35.

Note The system time of the NAQSServer computer must be synchronized with the Universal Time Coordinated (UTC) standard.

1.2 Summary of Inputs and Outputs

1.2.1 Required Input Files

NAQSServer requires two configuration files to be present in the current directory:

- `Naqs.ini` – Defines basic operating characteristics for NAQSServer (for example, IP addresses, ports).
- `Naqs.stn` – Defines stations, channels, instruments, triggers, and interconnections for this network.

See also Chapter 2 “Configuring NAQSServer”.

1.2.2 Output Files

1.2.2.1 Log File

The `Naqs_yyyymmdd.log` file contains diagnostic messages generated by NAQSServer and its associated instruments. The log provides a summary of the system operation. Each log message indicates the time and the source (instrument serial number or NAQS). A new log file is opened each day. The date is encoded in the log file name (for example, `Naqs_20031114.log`). Each log message has an associated type, ranked by severity (Table 1-1).

Log verbosity can be configured to show only messages at or above a specified severity level by adjusting the verbosity setting. The verbosity of the log on startup is set in the [NaqsLog] section of the `Naqs.ini` file (see Section 2.2.1 on page 13). During operation, verbosity can be set to a different level by using the runtime commands (see Section 3.2 on page 34).

Table 1-1 NAQSServer Log Message Types

Label	Description
F	Fatal errors – Serious errors which cause immediate system shutdown.
E	Errors – Abnormal occurrences which will likely affect data integrity.
W	Warnings – Less serious abnormal occurrences
I	Informational messages – Messages tracing the normal operation of the system.
V	Verbose messages – Detailed informational messages tracing the normal operation of the system.
D	Debug messages – Additional verbose trace messages.

1.2.2.2 Instrument Address File

The `naqsaddr.ini` file contains a table of the IP address and UDP port from which NAQSServer last received data for each instrument. This may be used either manually or by other software to determine the address through which to communicate with each instrument.

1.2.2.3 Event File

The `naqs.elf` file contains a list of seismic events detected by NAQSServer, including the start time and duration. This can be used in conjunction with the data-extraction utility (extractp) to extract event-related data from the ringbuffers. A new event file is opened each day. The date is encoded in the event file name (for example, `naqs_20031112.elf`). Triggers may optionally be written to the event file.

1.2.2.4 Ringbuffer Files

Ringbuffer filenames are automatically generated from the station and channel names specified in the configuration, using this formula:

```
filename = “R”
        + Station-Name (5 characters)
        + “.”
        + Channel-Name (3 characters)
```

Note The maximum number of ringbuffer files (SOH or data) that one instance of NAQSServer can support is 500.

1.2.2.4.1 Seismic Data Ringbuffers

Time-series data for each seismic channel are stored in ringbuffers in the original data-packet format.

Data may be extracted from the ringbuffers into Nanometrics X-file format using the extractp utility or the ExtractServer online extraction service. Nanometrics supports several data conversion utilities to convert X-files to Y-files (suitable for analysis by Atlas) or to various industry-standard formats such as SEED. Data may be extracted online, without shutting down NAQSServer.

1.2.2.4.2 Serial Data Ringbuffers

Serial data for each transparent serial channel are stored in ringbuffers in the original data-packet format. The ringbuffer filename is determined using the naming convention described below.

Data may be extracted from the ringbuffers using the SerialExtract utility. This removes all packet headers and delimiters and creates binary files which are suitable for post-processing by a domain-specific analysis program. Data may be extracted online while NAQSServer is running.

1.2.2.4.3 State-of-Health Ringbuffers

Each Nanometrics instrument provides numerous state-of-health (SOH) outputs detailing environmental and instrument status (for example, battery voltage, internal temperature, GPS location, timing accuracy), as well as three external state-of-health channels which are available for customer-specific measurements. SOH data for each instrument are stored in a separate ringbuffer in the original data-packet format.

SOH data may be extracted from the ringbuffers into comma-delimited text files (suitable for input to a spreadsheet program) using the Sohextrp utility or the ExtractServer online extraction service. SOH data may be extracted into Y-file format using the Sohtoyp utility. Data can be extracted online, without shutting down NAQSServer.

1.3 Installing NAQSServer

1.3.1 Software Requirements

Before you install NAQSServer, ensure that the installation computer meets the following requirements:

- Linux (kernel version 2.4 or later), Solaris 8, or Windows XP (Service Pack 2) operating system
- Java Runtime Environment version 1.6 or later

1.3.2 Upgrade Considerations

If you are upgrading from an earlier version of NAQSServer, ensure that you backup any user files that you have customized so that you can continue to use them with the new version. It is recommended that you back up the following files from the working directory /nmx/user before you follow the installation instructions:

- `Naqs.ini`
- `Naqs.stn`

1.3.3 Installing NAQSServer on Linux

1. Log in as root.
2. Copy all files from the bin folder on the installation CD to /nmx/bin.
   ```
   cd /mnt/cdrom/Software/Linux/NAQSServer
   cp bin/* nmx/bin/
   cp bin/.run* /nmx/bin/
   ```
3. Copy all files from the user folder on the installation CD to /nmx/user.
   ```
   cp user/* /nmx/user/
   ```
4. Install the required library files (.so).
   ```
   cp /mnt/cdrom/Software/Linux/LinuxLibraries/*/bin/* /nmx/bin/
   ldconfig
   ```
5. Set ownership of all Nanometrics files to nmx.
   ```
   cd /
   chown -R nmx:nmx /nmx
   ```
6. Set all files, except jar files, in /nmx/bin to executable.
   ```
   cd /nmx/bin
   chmod +x *
   chmod +x .run*
   chmod -x *jar
   ```
7. Add /nmx/bin to the system environment variable PATH.
8. Log off as root.
9. Copy the licence file `NaqsServer.lic` into the /nmx/user folder and insert the dongle into a USB port.

   Note If you do not have a licence file or USB dongle, contact a Nanometrics sales representative at sales@nanometrics.ca. Some Linux distributions might require a computer ID based licence instead of a USB dongle.

10. Set the configuration parameters as outlined in Chapter 2 “Configuring NAQSServer”.

1.3.4 Installing NAQSServer on Solaris

1. Log in as root.
2. Copy all files from the bin folder on the installation CD to /nmx/bin.
   ```
   cd /mnt/cdrom/Software/Solaris/NAQSServer
   cp bin/* /nmx/bin/
   cp bin/.run* /nmx/bin/
   ```
3. Copy all files from the user folder on the installation CD to /nmx/user.
   ```
   cp user/* /nmx/user/
   ```
4. Copy all Solaris library files from the installation CD into /usr/lib.
   ```
   cp /mnt/cdrom/Software/Solaris/SolarisLibraries/*/bin/* /nmx/bin/
   ```
5. Set ownership of all Nanometrics files to nmx.
   ```
   cd /
   chown -R nmx:nmx /nmx
   ```
6. Set all files, except jar files, in /nmx/bin to executable.
   ```
   cd /nmx/bin
   chmod +x *
   chmod +x .run*
   chmod -x *jar
   ```
7. Add /nmx/bin to the system environment variable PATH.
8. Log off as root.
9. Copy the licence file `NaqsServer.lic` into the /nmx/user folder.

   Note If you do not have a licence file, contact a Nanometrics sales representative at sales@nanometrics.ca.

10. Set the configuration parameters as outlined in Chapter 2 “Configuring NAQSServer”.

1.3.5 Installing NAQSServer on Windows

1. From either a command prompt or Microsoft Windows Explorer, open the installation CD folder Software\Win32\NAQSServer.
2. Copy all files from the bin folder into c:\nmx\bin.
3. Copy all files from the user folder into c:\nmx\user.
4. Open the installation CD folder Software\Win32\DLL and copy all files from the DLL folder into c:\nmx\bin.
5. Copy the licence file `NaqsServer.lic` into the c:\nmx\user folder and insert the dongle into a USB port.

Note If you do not have a licence file or USB dongle, contact a Nanometrics sales representative at sales@nanometrics.ca.

6. Add the folder \nmx\bin to the system environment variable PATH.
   - You can view and edit system environment variables by right-clicking My Computer and selecting Properties, clicking the Advanced tab, and clicking Environment Variables at the bottom of the Advanced dialog box. The system environment variables are shown in the lower pane of the Environment Variables dialog box.
7. Set the configuration parameters as outlined in Chapter 2 “Configuring NAQSServer”.

Chapter 2
Configuring NAQSServer

NAQSServer configuration is defined by two configuration files: The system configuration file `Naqs.ini` specifies general operating settings such as port numbers for TCP and UDP connections. The station configuration file `Naqs.stn` provides information about the instruments, stations, and channels that comprise the network. Both files are simple text files which can be created and modified using any text editor.

You must edit the two configuration files to instruct NAQSServer how to operate for your particular network. The main purposes of the configuration files are to:

- Provide the name and location of each station connected to the network.
- Define the name and characteristics of each data channel to be recorded.
- Provide a mapping between specific instruments and station/channel named data streams and files. This is done by specifying the instrument serial number and channel number for each named data channel.
- Specify other settings such as file sizes and trigger-detection parameters.

An overview of the configuration file structure is provided in Overview of Configuration File Structure (p. 9). Configuration parameters are defined in Section 2.2 “Naqs.ini File Parameters” on page 12 and Section 2.3 “Naqs.stn File Parameters” on page 17. Small example configuration files are shown in Appendix A.

2.1 Overview of Configuration File Structure

2.1.1 General Structure

Both `Naqs.ini` and `Naqs.stn` are structured as inifiles, a format which is designed to be readable, and editable in any text editor.

These files consist of a number of sections, each containing several parameters:

- Sections are identified by a name enclosed in square brackets (for example, [Network]).
- Each parameter is given on a separate line following the section identifier, in the format *ParameterName* = *Value*.

For example, a `Naqs.stn` file section defining a station location:

```
[ Station ]
Name = STN01
Description = New vault, bedrock, top of hill
Latitude = 45.31
Longitude = 18.37
Elevation = 1047.5
```

2.1.1.1 Data Order and Default Values

All parameters for a given section must appear after the section identifier for that section, and before any other section identifier.

Except where specified in the parameter descriptions, NAQSServer does not provide any default settings. Therefore, you must define fully every parameter in each section. Parameters must be defined in the order that they are listed in the parameter description sections. See Section 2.2 “Naqs.ini File Parameters” on page 12 and Section 2.3 “Naqs.stn File Parameters” on page 17.

2.1.1.2 Repeated Sections

Repeated sections are not permitted in the `Naqs.ini` file, but are permitted in the `Naqs.stn` file:

- Each section in the `Naqs.ini` file must appear exactly once, in the order specified in Section 2.2 on page 12.
- The `Naqs.stn` file accepts multiple definitions for every section type except [Network]. Each section defines an instance of the specified type (station, instrument, or channel). You do not need to specify the number of stations, instruments, and channels in the network: you only have to provide a section for each instance, adding new sections as required.

2.1.1.3 White Space and Comments

The inifile reader ignores white space and blank lines, so you can add white space anywhere within a configuration file to improve readability.

The double slash // is a comment delimiter. You can add comments anywhere in a file—to add descriptive information, and to remove parameters or sections temporarily from the file.

For example:

```
// This is a full line comment
[ Station ]     // a comment may follow a section header
Name = STN01    // a comment may follow a parameter definition
```

2.1.1.4 Inifile Error Detection

NAQSServer parses the configuration files on startup. If it detects any errors (unrecognized fields or illegal values), it will print an error message and stop. Illegal values are values which are undefined or out of range for a particular parameter. (See Section 2.2 and Section 2.3 for information on the permitted values for each parameter.)

- ▶ To resume, fix the file using a text editor, then restart NAQSServer.

The most common cause of unrecognized fields are:

- Misspelled parameter names – Check the spelling carefully, and note that parameter names are case-sensitive.
- Missing names – If a parameter appears out of order or in the wrong section, it will not be recognized.
- Duplicated names – If a parameter name appears twice, the second instance will not be recognized.

2.1.2 Station File Structure

The `Naqs.stn` file defines the stations, instruments, and channels which make up the network, and any trigger-detectors to be used. A small example `Naqs.stn` file is shown in Appendix A.

2.1.2.1 Prototypes and Instances

To promote standardization and reduce redundant data within the station file, instruments and channels are defined using prototypes:

- A prototype defines a standard instrument or channel.
- Each instance—that is, each specific instrument or channel—must specify:
  - A prototype.
  - Any other information (for example, serial number) required to define a unique instance. The instance definition can include values for any parameters that differ from those in the prototype.

2.1.2.2 Stations, Instruments, and Channels

Associations between stations, instruments, and channels are determined by their position in the `Naqs.stn` file:

- Each instrument is associated with the immediately preceding station.
- Each channel is associated with the immediately preceding instrument and station.
- Associations between triggers and channels are determined by name; each trigger specifies the name of the channel to which it is to be applied.

There may be any number of instruments per station, and any number of channels per instrument, provided that the station-channel name is unique for each channel.

The structure of the `Naqs.stn` file is summarized in Figure 2-1. See Section 2.3 on page 17 for definitions of the file sections and parameters.

Figure 2-1 Naqs.stn file structure

[Section]	**// Description**
[Network]	// Main network parameters
// Repeat for each sensor in the network	
[Sensor]	// Sensor characteristics
// Repeat for each instrument prototype	
[InstrumentPrototype]	// Definition of a prototype instrument
// Repeat for each seismic channel prototype	
[ChannelPrototype]	// Definition of a prototype seismic channel
// Repeat for each serial channel prototype	
[SerialChannelPrototype]	// Definition of a prototype serial channel
// Repeat for each station S in the network	
[Station]	// Station name and location
// Repeat for each instrument I of station S	
[Instrument]	// Settings for instrument I
// Repeat for each channel C of instrument I	
[Channel]	// Name and settings for channel C
// End of channel C	
// Repeat for each serial channel C of instrument I	
[SerialChannel]	// Name and settings for serial channel C
// End of serial channel C	
// End of instrument I	
// End of station S	
// Repeat for each detector type in the network	
[DetectorType]	// Trigger detector filter and level definitions
// Repeat for each channel-specific trigger in the network	
[Trigger]	// Detector type and channel for a specific trigger

2.2 Naqs.ini File Parameters

The `Naqs.ini` file contains six sections, to configure the major NAQSServer subsystems. Parameters are mandatory unless indicated otherwise. Each section in the `Naqs.ini` file must appear exactly once, in this order:

- [NaqsLog]
- [AlertSender]
- [NetworkInterface]
- [EventAssociator]
- [Datastream]
- [Calibrator]

A small example `Naqs.ini` file is shown in Appendix A.

▸ If you edit to the `Naqs.ini` file, you must restart NAQSServer for the change to take effect.

2.2.1 [NaqsLog]

The [NaqsLog] section defines the location and name of the NAQSServer log file and its verbosity on startup. It contains the parameters defined in Table 2-1.

Table 2-1 [NaqsLog] Section Parameters

Parameter	Definition
LogPath	The pathname for the directory in which to store the NAQSServer log file. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `LogPath = data`
LogFile	The base name for the NAQSServer log file. NAQSServer creates a new log file every day. The log file name is determined by inserting the date (*yyyymmdd*) between the base name and the file extension; for example, `Naqs_20031114.log`. ◆ Permitted values: Any valid file name, with no spaces. Example: `LogFile = Naqs.log`
Verbosity	The startup verbosity of the NAQSServer log file. ◆ Permitted values: DEBUG, VERBOSE, INFO. Example: `Verbosity = INFO`

2.2.2 [AlertSender]

The [AlertSender] section defines the destination for alert messages generated by NAQSServer. It contains the parameters defined in Table 2-2. For further information on alert messages, see the AlertMailer user guide.

Table 2-2 [AlertSender] Section Parameters

Parameter	Definition
InetHost	The IP address of the computer running the alert forwarding software being used. This may be either a unicast address or a multicast address. Specify “none” to disable sending of alert messages. ◆ Permitted values: Any valid unicast or multicast IP address, or “none”. Example: `InetHost = none`
Port	The UDP port to which to send alert messages. ◆ Permitted values: Any valid port number (typically 31000). Example: `Port = 31000`
SourceID	The name by which alert messages from this program will be identified. This allows message recipients to determine which program or instrument issued a given alert message. ◆ Permitted values: Any string. Example: `SourceID = Naqs-A`

2.2.3 [NetworkInterface]

The [NetworkInterface] section configures the network interface module, which is responsible for communication with all data acquisition instruments. It contains the parameters listed in Table 2-3, and may contain the optional parameters listed in Table 2-4.

Table 2-3 [NetworkInterface] Section Parameters

Parameter	Definition
Port	The UDP port used for communication with data acquisition instruments. Nanometrics comms devices should be configured to send inbound data to this port. ◆ Permitted values: A positive integer (typically 32000). Example: `Port = 32000`
SendDelay	The delay in milliseconds after sending each outbound packet (for example, retransmission requests). This should normally be zero; non-zero values may be used to control the transmission rate to eliminate packet losses over a limited-bandwidth outbound link. If all retransmission requests are to be send over a single 9600 baud radio link, *SendDelay* should be set to 250 (4 packets per second). ◆ Permitted values: Integer, 0 to 1000. Example: `SendDelay = 250`

Table 2-4 Optional Parameters for [NetworkInterface]

Parameter	Definition
RetxRequest	Enable or disable NAQSServer to send out retransmission requests for missed packets. If this parameter is omitted, the NAQSServer default is to enable sending of retransmission request messages. ◆ Permitted values: Enabled, Disabled. Example: `RetxRequest = Enabled`
MulticastGroup	The IP address of a multicast group on which the NAQSServer can receive messages. NAQSServer can listen to any number of multicast groups; include one line in the configuration file for each group. If no multicast groups are specified, NAQSServer will still receive data packets sent via unicast UDP to the correct computer and port. The IP address of a multicast group to which NAQSServer should subscribe to receive data. NAQSServer may subscribe to multiple multicast groups; include one line for each group. NAQSServer will receive multicast packets sent to the correct port number on any of its subscribed groups, as well as unicast packets sent to the IP address of the NAQSServer computer. Remove this line if NAQSServer should not receive from any multicast groups. ◆ Permitted values: Any valid multicast IP address. Example: `MulticastGroup = 225.1.1.1`

2.2.4 [EventAssociator]

The [EventAssociator] section configures the event associator module, which is responsible for associating seismic trigger packets to detect seismic events. It contains the parameters defined in Table 2-5.

Table 2-5 [EventAssociator]Section Parameters

Parameter	**Definition**
EventPath	The pathname for the directory in which to store the NAQSServer event file. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `EventPath = data`
EventFile	The base name of the NAQSServer event file. A new event file is created every day. The event file name is determined by inserting the date (*yyyymmdd*) between the base name and the file extension; for example, `Naqs_20031114.elf`. ◆ Permitted values: Any valid file name, with no spaces. Example: `EventFile = Naqs.elf`
TrigsToStart	The number of data channels that must be triggered within the Coincidence Window in order to declare the start of an event. ◆ Permitted values: Integer, 1 to 100. Example: `TrigsToStart = 3`
CoincidenceWindow	The width, in seconds, of the time window into which the trigger-on times of data channels must fall in order for those channels to be included in the same event. ◆ Permitted values: Integer, 0 to 3600. Example: `CoincidenceWindow = 20`
EventTimeout	The maximum duration of an event. This is the maximum amount of time to defer sending an event packet, waiting for its active channels to stop triggering. ◆ Permitted values: Integer, 0 to 3600. Example: `EventTimeout = 300`
SaveTriggers	Write triggers to the event file. ◆ Permitted values: Yes, No. Example: `SaveTriggers = Yes`

2.2.5 [Datastream]

The [Datastream] section configures the Naqs Datastream service, which is responsible for sending private data streams to subscribing client programs such as Waveform and NaqsView. It contains the parameters defined in Table 2-6.

Table 2-6 [Datastream] Section Parameters

Parameter	Definition
Port	The TCP port used for the data subscription service. This is configurable to avoid conflicts with other services, and to meet firewall requirements. ◆ Permitted values: Any available port number. Example: `Port = 28000`
Password	Subscription password. ◆ Permitted values: Any string. Example: `Password = none`
MaxConnections	The maximum number of data subscriptions permitted. This can be used to limit access to the NAQSServer data streams. ◆ Permitted values: Integer, 0 to 20 Example: `MaxConnections = 12`
SocketType	The socket type to use for connection to data-subscription clients. NAQSServer acts as a data server: NAQSServer listens passively for TCP connections from client programs; client programs actively attempt to connect to NAQSServer. Once a connection is established, NAQSServer communicates with a specific client using the protocols defined in the DataStreams section. NAQSServer supports two different connection methods—direct and callback: ◆ In the direct method, communication between NAQSServer and the client is carried over the original connection established by the client. ◆ In the callback method, the original connection is used only to request a callback from NAQSServer. NAQSServer closes the first connection, then opens a second connection to the client (that is, calls back); all communication between NAQSServer and the client is carried on the second connection. Most installations should use the direct method; some installations with firewalls may have to use the callback method. ◆ Permitted values: Direct or Callback. Example: `SocketType = Direct`
DataBufferLength	The length in packets of the pre-event buffer for each data channel. The pre-event buffer for a channel is (optionally) sent when a subscription is opened for that channel. This allows a data stream client to operate in event-driven mode, in which time-series data are requested only after an event message is received. ◆ Permitted values: Integer, 0 to 100 (in packets) Example: `DataBufferLength = 30`

2.2.6 [Calibrator]

The [Calibrator] section configures the calibration module, which accepts calibration and mass-centering requests from client programs such as Calibrate and NaqsView. It contains the parameters defined in Table 2-7.

Table 2-7 [Calibrator] Section Parameter

Parameter	Definition
Password	Calibration password. Use of a password is recommended to prohibit calibration and mass-centring commands from being run by unauthorized persons. ◆ Permitted values: Any string. Example: `Password = Calpass`

2.3 Naqs.stn File Parameters

The `Naqs.stn` file contains a section to identify the network, and sections to configure each of the stations, instruments, channels, and triggers. Except for the single [Network] section, there may be multiple sections of each type:

- [Network]
- [Sensor]
- [InstrumentPrototype]
- [ChannelPrototype]
- [SerialChannelPrototype]
- [Station]
- [Instrument]
- [Channel]
- [SerialChannel]
- [DetectorType]
- [Trigger]

For an overview of station file structure, see Section 2.1.2 on page 11 and Figure 2-1 on page 12. A small example `Naqs.stn` file is shown in Appendix A.

- If you edit the `Naqs.stn` file, you must restart NAQSServer for the change to take effect.

2.3.1 [Network]

The [Network] section contains settings which apply to the network as a whole. It defines the network name and identifies the acquisition system. There can be only one [Network] section within the file. It contains the parameters defined in Table 2-8.

Table 2-8 [Network] Section Parameters

Parameter	Definition
Name	The name of the network. The first 3 characters of this appear in each log message generated by NAQSServer. The network name is also written into the NetworkID field in the header of each ringbuffer. ◆ Permitted values: Any character string. Example: `Name = ETH`
NaqsId	Acquisition system identifier. This is used to distinguish among two or more acquisition systems serving the same network. The *NaqsId* appears in every log message generated by NAQSServer. ◆ Permitted values: Any single character. Example: `NaqsId = A`

2.3.2 [Sensor]

The [Sensor] section defines the characteristics for a specific sensor type used within the network. Sensor characteristics are written to the ringbuffer files for each channel. Calibration and mass-centring characteristics are specified here to allow NAQSServer to perform calibration and mass centering correctly for each sensor type. There can be any number of [Sensor] sections in the station file. It contains the parameters defined in Table 2-9.

- Create a separate section for each sensor type.
- Group these definitions near the top of the station file before the definition of any ChannelPrototypes or Channels.

Table 2-9 [Sensor] Section Parameters

Parameter	Definition
TypeName	The name of this sensor prototype. ChannelPrototypes and Channels refer to a sensor using its *TypeName.* ◆ Permitted values: Any character string. Example: `TypeName = STS-2`
Model	The sensor model name. This name is written to the SensorType field in the Y-File header of the data ringbuffers. ◆ Permitted values: Any character string. Example: `Model = STS-2`
SensitivityUnits	Units of ground motion (m, m/s or m/s^2). This is written to the SensitivityUnits field in the Y-File header of the data ringbuffers. ◆ Permitted values: M, M/S, or M/S**2. Example: `SensitivityUnits = M/S`
Sensitivity	Data counts per unit of ground motion (typically 1.0e+9 counts per m/s). This is written to the Y-File header of the ringbuffers. ◆ Permitted values: Any floating point value. Example: `Sensitivity = 1.0e+9`

Table 2-9 [Sensor] Section Parameters (Continued)

Parameter	Definition
SensitivityFreq	Frequency in hertz at which the Sensitivity is correct. This is written to the Y-File header of the ringbuffers. ◆ Permitted values: Any positive floating point value. Example: `SensitivityFreq = 1.0`
CalibrationUnits	Calibration input units for this sensor. This is written to the Y-File header of the ringbuffers, and used to compute the required signal output during calibration. ◆ Permitted values: VOLTS or AMPS. Example: `CalibrationUnits = VOLTS`
CalCoilResistance	Calibration coil resistance in ohms. This is used to compute the required signal amplitude during calibration. ◆ Permitted values: Any positive floating point value. Example: `CalCoilResistance = 20000`
CalCoilConstant	Calibration coil constant, expressed as calibration units per unit acceleration (for example, volts per m/s/s or amps per m/s/s). This is used to compute the required signal amplitude during calibration. Some sensor manufacturers may express the coil constant in different units: ▸ The constant must always be converted to the correct units in order for the NAQSServer calibration feature to work correctly. For example, if your coil constant is expressed as N/volt, divide by the sensor mass in kg, then invert to get the constant in volts per m/s/s. ◆ Permitted values: Any positive floating point value. Example: `CalCoilConstant = 33.2`
CalEnable	This indicates which digital control line(s) of the digitiser should be set active to enable calibration on this sensor. This depends on both the sensor and the sensor cable, and will be specified in the cable drawings provided by Nanometrics. This value must be specified correctly in order for the NAQSServer calibration feature to work correctly. The digitiser has 3 digital control lines. ◆ Permitted values: • –1 No digital control line is required to enable calibration. Use this option for passive seismometers or for seismometers which are enabled via the (CAL –) relay lines.* *Specify CalEnable = –1 and CalRelay = –1 for sensors that do not support calibration. • 0 Channels on this sensor may be calibrated individually; calibration of each channel is enabled by a different control line (for example, control 1 enables channel 1). • 1 to 3 All channels on this sensor are enabled using the specified control line (1 to 3) Example: `CalEnable = 2`

Table 2-9 [Sensor] Section Parameters (Continued)

Parameter	Definition
CalRelay	This indicates which calibration relay(s) on the digitiser should be closed to calibrate this sensor. Each digitiser has 6 double-pole relays. The positive pole of each relay is typically used to carry a sinusoidal analog calibration signal from the digitiser to the sensor coil. The negative pole of each relay can be used to enable calibration on a specific channel. Relay assignment depends on both the sensor and the sensor cable, and will be specified in the cable drawings provided by Nanometrics. The correct relay value must be specified here in order for the NAQSServer calibration feature to work correctly. ◆ Permitted values: • –1 Sinusoidal calibration is not supported for this sensor. • 0 Each channel on this sensor has a separate calibration coil; each coil is connected to a separate relay (for example, channel 1 to relay 1). This allows each channel to be calibrated individually. • 1 to 6 The coil(s) for all channels are connected to the specified relay. This would typically be used for a sensor which does not support calibration of individual channels. • 8 This sensor has a single calibration coil, but separate calibration enables for each channel. The positive pole of each relay (CAL+) is connected to the coil; the negative pole of each relay (CAL–) is connected to the calibration enable for the corresponding channel. This allows each channel to be calibrated individually. Example: `CalRelay = 0`
MassCenterEnable	This indicates which digital control line(s) of the digitiser should be set active to initiate mass centring on this sensor. This depends on both the sensor and the sensor cable, and will be specified in the cable drawings provided by Nanometrics. The correct value must be specified here in order for the NAQSServer mass centring feature to work correctly. A Nanometrics digitiser has 3 digital control lines. ◆ Permitted values: • –1 Mass centering is not supported for this sensor. • 0 Channels on this sensor may be centred individually; mass centring of each channel is initiated using a different control line (for example, control 1 for channel 1). • 1 to 3 All channels on this sensor must be centred simultaneously using the specified control line (1 to 3). Example: `MassCenterEnable = 1`
MassCenterDuration	Gives the duration in seconds of the mass-centring signal required for this sensor. The default value (if this parameter is omitted) is 1 second. ◆ Permitted values: Any positive integer. Example: `MassCenterDuration = 5`
CalSource	Specifies the digitiser type through which the sensor will be calibrated. The parameter is required in order to allow Naqs to correctly compute the calibration signal to be sent to the digitiser. ◆ Permitted values: Trident, HRD. Example: `CalSource = Trident`

2.3.3 [InstrumentPrototype]

The [InstrumentPrototype] section defines the characteristics for a specific type of instrument used in the network. An instrument prototype defines a “standard” instrument; each specific instrument must refer to a prototype, but may override one or more parameters. It contains the parameters defined in Table 2-10.

There can be any number of [InstrumentPrototype] sections within a station file.

- Group these definitions together near the top of the station file before any Station definitions.

Table 2-10 [InstrumentPrototype] Section Parameters

Parameter	Definition
TypeName	The name of this instrument prototype. Instruments refer to a prototype using its *TypeName.* ◆ Permitted values: Any character string. Example: `TypeName = Trident`
Model	The instrument model. ◆ Permitted values (values are case sensitive): • HRD • Orion • RM3 • RM4 • LYNX • Cygnus • Cygnus205 • Carina • Carina105 • Europa • Janus • Trident • Trident305 • Taurus Example: `Model = Trident`
MemoryKB	The onboard memory of the instrument in kilobytes. This value is used to limit the space reserved for retransmission of lost data packets. If retransmission is not enabled this should be set to zero, otherwise it should be set to the actual memory size of the digitiser (typically 12MB). ◆ Permitted values: Integer greater than 0. Example: `MemoryKB = 4000`
SohBundlesPerPacket	The number of 17-byte data bundles per data packet. The parameter value is required to define the correct internal structure for the ringbuffers. This value must be set to the same value as that used by the digitisers attached to the network (refer to the as-shipped configuration sheets) for all of the instrument models except for Trident, Trident305, Taurus, Cygnus205, and Carina105. ◆ Permitted values: Any odd integer from 1 to 59. • The parameter value must be set to 15 for Trident and Trident305. • The parameter value must be set to 27 for Taurus, Cygnus205, and Carina105. Example: `SohBundlesPerPacket = 15`

Table 2-10 [InstrumentPrototype] Section Parameters (Continued)

Parameter	Definition
RequestInterval	The re-request interval in seconds. Missing data is requested repeatedly until it is received or until it is no longer available for retransmission. This parameter specifies how often the retransmission requested are repeated. ◆ Permitted values: Integer from 30 to 1200. Example: `RequestInterval = 300`
SohChannelName	The channel name for the multiplexed state-of-health data for this instrument. This name is used as the extension for the state-of-health ringbuffer file name, and written to the Channel field of the SEED station ID structure in the ringbuffer. ◆ Permitted values: • Any 3-character string. All letters in the name must be uppercase. • For stations with multiple instruments, each instrument must have a unique *SohChannelName*. Example: `SohChannelName = SOH`
SohBufferSize	The size in megabytes of the state-of-health ringbuffer for each instrument. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `SohBufferSize = 1`
SohBufferPath	Pathname for the directory in which to store the SOH ringbuffer files for each instrument derived from this prototype. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname. Example: `SohBufferPath = ringbuffer`
InetHostName	The IP address of the instrument expressed either in dotted decimal format, or as an IP host name if one has been defined. For a Nanometrics comms device, this is the Internet address of the unit. For associated Tridents or internal HRDs, this is the IP address of the comms device to which the digitiser is connected. This field provides a return address for the instrument. If you use the value "Dynamic" for this field rather than the IP address, the return address is updated dynamically whenever a packet is received from an instrument. ◆ Permitted values: Any valid IP address or host name, or Dynamic. Example: `InetHostName = Dynamic`
InetPort	The UDP port number used for outbound communication to this instrument. This field provides part of the return address for this instrument (see also *InetHostName*, above). This must match the configuration on the instrument. If *InetHostName* is set to Dynamic, this field is updated dynamically whenever a packet is received from an instrument. ◆ Permitted values: A positive integer. Example: `InetPort = 32000`

2.3.4 [ChannelPrototype]

The [ChannelPrototype] section defines the characteristics for a "standard" channel of time-series data. It is a convenient place to define parameters which are common to many channels throughout the network. Each specific channel defined later must refer to a prototype, but may override certain parameters.

There may be any number of [ChannelPrototype] sections within a station file. A simple network might have three channel prototypes (Z, N, and E); a more complex network with a mix of sensor types would have more. It contains the parameters defined in Table 2-11.

- Group these definitions together near the top of the station file following the Sensor definitions, but before any Station definitions.

Table 2-11 [ChannelPrototype] Section Parameters

Parameter	**Definition**
TypeName	The name of this channel prototype. Channels refer to a prototype using its *TypeName*. ◆ Permitted values: Any character string. Example: `TypeName = BHZ-1`
Name	The channel name. The first 3 characters of this name are encoded as the extension of the ringbuffer file names, and written to the Channel field of the SEED station ID structure in the ringbuffer. ◆ Permitted values: Any character string. All letters in the name must be uppercase. Example: `Name = BHZ`
Component	The digitiser source channel for this time series. ◆ Permitted values: Integer from 1 to 6. Example: `Component = 1`
Sensor	The *TypeName* of the sensor for this channel. The sensor must have been previously defined by a [Sensor] section. ◆ Permitted values: Any valid sensor TypeName. Example: `Sensor = STS-2`
Azimuth	The azimuth angle (degrees) of the active axis for this channel. ◆ Permitted values: –180.0 to 180.0. Example: `Azimuth = 0`
Dip	The dip angle (degrees) of the active axis for this channel. ◆ Permitted values: –90.0 to 90.0. Example: `Dip = 90`
Depth	The depth of the sensor (metres) for this channel. ◆ Permitted values: Any floating point value. Example: `Depth = 0`
BundlesPerPacket	The number of 17-byte data bundles per data packet. The parameter value is required to define the correct internal structure for the ringbuffers. This value must be set to the same value as that used by the digitisers attached to the network (refer to the as-shipped configuration sheets) for all of the instrument models except for Trident, Trident305, Taurus, Cygnus205, and Carina105. ◆ Permitted values: Any odd integer from 1 to 59. • The parameter value must be set to 15 for Trident and Trident305. • The parameter value must be set to 27 for Taurus, Cygnus205, and Carina105. Example: `SohBundlesPerPacket = 15`
RingBufferSize	The size in megabytes of the data ringbuffer for this channel. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `RingBufferSize = 10`

Table 2-11 [ChannelPrototype] Section Parameters (Continued)

Parameter	Definition
RingBufferPath	Pathname for the directory in which to store the data ringbuffer files for each channel derived from this prototype. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `RingBufferPath = ringbuffer`
ResponseFile	The name of the response file name for this channel. This is written to the Y-File header of the ringbuffers, and is used by makeseed when constructing a SEED volume from extracted data. ◆ Permitted values: Any valid filename, with no spaces. Example: `ResponseFile = none`

2.3.5 [SerialChannelPrototype]

NAQSServer can receive and archive arbitrary channels of binary serial data received on a serial port by a Nanometrics comms device. Data are stored to ringbuffers and may be extracted by off-line programs. Incoming data are identified by the source instrument and port, and are written to ringbuffer files identified by a station and channel name.

The [SerialChannelPrototype] section defines the characteristics for a “standard” channel of serial data. It is a convenient place to define parameters which are common to many channels throughout the network. Each specific serial-data channel defined later must refer to a prototype, but may override certain parameters. It contains the parameters defined in Table 2-12.

There may be any number of [SerialChannelPrototype] sections within a station file.

- Group these definitions together near the top of the station file before any Station definitions.

Table 2-12 [SerialChannelPrototype] Section Parameters

Parameter	Definition
TypeName	The name of this channel prototype. SerialChannels refer to a prototype using its *TypeName*. ◆ Permitted values: Any character string. Example: `TypeName = GPS`
Name	The channel name. The first 3 characters of this name are encoded as the extension of the ringbuffer file names, and written to the Channel field of the SEED station ID structure in the ringbuffer. ◆ Permitted values: Any character string. All letters in the name must be uppercase. Example: `Name = GPS`
Description	A character string describing the type or source of data being recorded. ◆ Permitted values: Any character string. Example: `Description = wind speed`
Port	The Nanometrics comms device source port for this data channel. ◆ Permitted values: Integer, 1 to 8. Example: `Port = 3`

Table 2-12 [SerialChannelPrototype] Section Parameters (Continued)

Parameter	Definition
BytesPerPacket	The maximum number of data bytes per packet. This must be set to the same value as that used by the instrument receiving the data via serial port. The value is required here in order to define the correct internal structure for the ringbuffers. ◆ Permitted values: 17 * N, where N is any odd integer from 1 to 59. Example: `BytesPerPacket = 255`
RingBufferSize	The size in megabytes of the data ringbuffer for this channel. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `RingBufferSize = 10`
RingBufferPath	Pathname for the directory in which to store the ringbuffer files for each channel derived from this prototype. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `RingBufferPath = ringbuffer`

2.3.6 [Station]

The [Station] section defines the name and location of a specific seismic station. Each station should be followed immediately by one or more [Instrument] sections defining the instrument(s) located at that station. It contains the parameters defined in Table 2-13.

- Create a separate section for each station in the network.

Table 2-13 [Station] Section Parameters

Parameter	Definition
Name	The station name. The first five characters of this name are encoded in the ringbuffer file names, and written to the Station field of the SEED station ID structure in the ringbuffers. ◆ Permitted values: Any character string. All letters in the name must be uppercase. Example: `Name = STN05`
Description	A descriptive comment to be written to the Y-File header for ringbuffers associated with this station. ◆ Permitted values: Any character string. Example: `Description = none`
Latitude	The latitude of the station in degrees. This value is recorded in the data and state-of-health ringbuffers. ◆ Permitted values: –90.0 to 90.0. Example: `Latitude = 48.02`
Longitude	The longitude of the station in degrees. This value is recorded in the data and state-of-health ringbuffers. ◆ Permitted values: –180.0 to 180.0. Example: `Longitude = 14.96`
Elevation	The elevation of the station in metres. This value is recorded in the data and state-of-health ringbuffers. ◆ Permitted values: –9999.0 to 9999.0. Example: `Elevation = 0.00`

2.3.7 [Instrument]

The [Instrument] section defines the characteristics for a specific instrument. It contains the parameters listed in Table 2-14, and may contain the optional parameters listed in Table 2-15.

A station may have any number of instruments, provided that each SohChannelName is unique. The [Instrument] section for each instrument must be followed immediately by the [Channel] sections defining its seismic channels, and / or the [SerialChannel] sections defining its serial channels.

Each instrument is derived from an instrument prototype. To fully define a unique instrument, you must specify the prototype name and the instrument serial number. The instrument then inherits all of its characteristics from the prototype. Parameters which differ from the prototype can be set using the optional fields.

- Create a separate section for each instrument.
- Insert the section following the [Station] section for the station to which it belongs.

Table 2-14 [Instrument] Section Parameters

Parameter	Definition
Prototype	The *TypeName* of the instrument prototype which defines the default characteristics for this instrument. The prototype must have been previously defined by an [InstrumentPrototype] section. ◆ Permitted values: Any valid instrument prototype *TypeName*. Example: `Prototype = HRD`
SerialNumber	The instrument serial number. The instrument type and serial number must be correct in order to receive data from this instrument. Data packets are encoded with the serial number of the source instrument; NAQSServer uses serial number to direct packets to the correct files. ◆ Permitted values: Integer from 0 to 2047. Example: `SerialNumber = 201`

Typically, an instrument inherits all of its characteristics from its prototype. Parameters which differ from the prototype may be set using the optional fields described in Table 2-15. It is only necessary to specify parameters which differ from those of the prototype; if the instrument is identical to the prototype, all optional fields can be omitted. If an instrument differs from the prototype in more than one or two parameters, it may be preferable to define another prototype.

All parameters are optional. Included parameters must be specified in the same order as they are listed here.

Table 2-15 Optional Parameters for [Instrument]

Parameter	Definition
MemoryKB	The onboard memory of the instrument in kilobytes. This value is used to limit the space reserved for retransmission of lost data packets. If retransmission is not enabled, this should be set to zero; otherwise it should be set to the actual memory size of the digitiser (typically 12MB). ◆ Permitted values: Integer greater than 0. Example: `MemoryKB = 4000`
SohBundlesPerPacket	The number of 17-byte data bundles per data packet. This must be set to the same value as that used by the digitisers attached to the network (refer to the as-shipped configuration sheets). The value is required here in order to define the correct internal structure for the ringbuffers. ◆ Permitted values: Any odd integer from 1 to 59. Example: `SohBundlesPerPacket = 15`
RequestInterval	The re-request interval in seconds. Missing data is requested repeatedly until it is received or until it is no longer available for retransmission. This parameter specifies how often the retransmission requested are repeated. ◆ Permitted values: Any integer from 30 to 1200. Example: `RequestInterval = 300`
SohChannelName	The channel name for the multiplexed state-of-health data for this instrument. This name is used as the extension for the state-of-health ringbuffer file name, and written to the Channel field of the SEED station ID structure in the ringbuffer. Use NUL to indicate that SOH data from this instrument should not be recorded under this station name. This allows a single instrument to be assigned to several stations without conflict. ◆ Permitted values: • Any 3-character string or NUL for none. All letters in the name must be uppercase. • For stations with multiple instruments, each instrument must have a unique *SohChannelName*. Example: `SohChannelName = SOH`
SohBufferSize	The size in megabytes of the state-of-health ringbuffer for this instrument. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `SohBufferSize = 1`
SohBufferPath	Pathname for the directory in which to store the Soh ringbuffer files for this instrument. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `SohBufferPath = RingBuff`
InetHost	The IP address of the instrument expressed either in dotted decimal format, or as an IP host name if one has been defined. For a Nanometrics comms device this is the Internet address of the unit. For associated Tridents or internal HRDs, this is the IP address of the comms device to which the digitiser is connected. This field provides a return address for the instrument. If you use the value “Dynamic” for this field, rather than the IP address, the return address is updated dynamically whenever a packet is received from an instrument. ◆ Permitted values: Any valid IP address or host name, or Dynamic. Example: `InetHost = Dynamic`

Table 2-15 Optional Parameters for [Instrument] (Continued)

Parameter	Definition
InetPort	The UDP port number used for outbound communication to this instrument. This field provides part of the return address for this instrument (see also *InetHost*, above). This must match the configuration on the remote instrument. If *InetHost* is specified as “Dynamic” this field is updated dynamically whenever a packet is received from an instrument. ◆ Permitted values: A positive integer. Example: `InetPort = 32000`

2.3.8 [Channel]

The [Channel] section defines the characteristics of a single seismic data channel. It contains the parameter listed in Table 2-16, and may contain the optional parameters listed in Table 2-17.

Each channel is derived from a channel prototype, therefore you must specify the prototype name. All channel characteristics are then inherited from the prototype. Parameters which differ from the prototype can be set using the optional fields.

- Create a separate section for each data channel.
- Insert Channel sections immediately after the [Instrument] section for the appropriate digitiser.

Table 2-16 [Channel] Section Parameter

Parameter	Definition
Prototype	The *TypeName* of the channel prototype which defines the default characteristics for this channel. The prototype must have been previously defined by a [ChannelPrototype] section. ◆ Permitted values: Any valid channel prototype *TypeName*. Example: `Prototype = BHZ-1`

Typically, a channel inherits all of its characteristics from the prototype. Parameters which differ from the prototype can be set using the optional fields described in Table 2-17. It is only necessary to specify parameters which differ from those of the prototype. If the channel is identical to the prototype, all optional fields can be omitted. If a channel differs from the prototype in more than two or three parameters, it may be preferable to define another prototype.

Note Channel name, component, and sensor are key fields of a channel prototype and cannot be overridden

All of the following parameters are optional. All parameters which are included must be specified in the same order as they are listed here.

Table 2-17 Optional Parameters for [Channel]

Parameter	Definition
Azimuth	The azimuth angle (degrees) of the active axis for this channel. ◆ Permitted values: –180.0 to 180.0. Example: `Azimuth = 0`
Dip	The dip angle (degrees) of the active axis for this channel. ◆ Permitted values: –90.0 to 90.0. Example: `Dip = 90`
Depth	The depth of the sensor (metres) for this channel. ◆ Permitted values: Any floating point value. Example: `Depth = 0`
RingBufferSize	The size in megabytes of the data ringbuffer for this channel. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `RingBufferSize = 10`
RingBufferPath	The pathname for the directory in which to store the data ringbuffer files for this channel. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname. Example: `RingBufferPath = ringbuffer`
ResponseFile	The name of the response file name for this channel. This is written to the Y-File header of the ringbuffers, and is used by makeseed when constructing a SEED volume from extracted data. ◆ Permitted values: Any valid filename. Example: `ResponseFile = none`
Sensitivity	Data counts per unit of ground motion (typically 1.0e+9 counts per m/s). This is written to the Y-File header of the ringbuffers. ◆ Permitted values: Any floating point value. Example: `Sensitivity = 1.0e+9`
SensitivityFreq	Frequency in hertz at which the above sensitivity is correct. This is written to the Y-File header of the ringbuffers. ◆ Permitted values: Any valid pathname. Example: SensitivityFreq = 1.0
CalCoilResistance	Calibration coil resistance in ohms. This is used to compute the required signal amplitude during calibration. ◆ Permitted values: Any positive floating point value. Example: `CalCoilResistance = 20000`
CalCoilConstant	Calibration coil constant, expressed as calibration units per unit acceleration (for example volts per m/s/s or amps per m/s/s). This is used to compute the calibration signal amplitude. Some sensor manufacturers may express the coil constant in different units: ▸ The constant must always be converted to the correct units in order for the NAQSServer calibration feature to work correctly. For example, if your coil constant is expressed as N/volt, divide by the sensor mass in kg, then invert to get the constant in volts per m/s/s. ◆ Permitted values: Any positive floating point value. Example: `CalCoilConstant = 33.2`

2.3.9 [SerialChannel]

NAQSServer can receive and archive arbitrary channels of binary serial data received on a serial port by a Nanometrics comms device. Data are stored to ringbuffers and may be extracted by offline programs. Incoming data are identified by the source instrument and port, but are written to ringbuffer files identified by a standard station and channel name.

The [SerialChannel] section defines the characteristics of a single channel of binary serial data. Each channel is derived from a channel prototype, therefore you must specify the prototype name. All channel characteristics are then inherited from the prototype. Parameters which differ from the prototype can be set using the optional fields. It contains the parameter described in Table 2-18, and may contain the optional parameters listed in Table 2-19.

SerialChannel sections should follow immediately after the [Instrument] section for the appropriate instrument. SerialChannel sections may appear either before or after Channel sections for the same instrument.

- Create a separate section for each channel.

Table 2-18 [SerialChannel] Section Parameter

Parameter	Description
Prototype	The *TypeName* of the serial channel prototype which defines the default characteristics for this channel. The prototype must have been previously defined by a [SerialChannelPrototype] section. ◆ Permitted values: Any valid serial channel prototype *TypeName*. Example: `Prototype = GPS`

Typically, a channel inherits all of its characteristics from the prototype. Parameters which differ from the prototype can be set using the optional fields described in Table 2-19. It is only necessary to specify parameters which differ from those of the prototype. If the channel is identical to the prototype, all optional fields may be omitted.

All of the following parameters are optional. All parameters which are included must be specified in the same order as they are listed here.

Table 2-19 Optional Parameters for [SerialChannel]

Parameter	Description
Port	The Nanometrics comms device source port for this data channel. ◆ Permitted values: Integer from 1 to 8. Example: `Port = 3`
RingBufferSize	The size in megabytes of the data ringbuffer for this channel. ◆ Permitted values: An integer less than or equal to 2000 MB. Example: `RingBufferSize = 10`
RingBufferPath	Pathname for the directory in which to store the ringbuffer files for this channel. Do not include the trailing slash. Names are treated as relative pathnames (relative to the directory in which NAQSServer is running), unless they are specified as absolute names (with a leading slash). ◆ Permitted values: Any valid pathname, with no spaces. Example: `RingBufferPath = ringbuffer`

2.3.10 [DetectorType]

The [DetectorType] section defines the filter characteristics for a specific named detector type. It contains the parameters defined in Table 2-20.

- Create a separate section for each named detector.

Table 2-20 [DetectorType] Section Parameters

Parameter	Description
Name	The name of this detector type. ◆ Permitted values: Any string. Each DetectorType *Name* must be unique. Example: `Name = Teleseism`
TypeID	An integer used as a short identifier for this detector type. ◆ Permitted values: Any unique integer. Example: `TypeID = 2`
CompletionDelay	The short-term-completion delay in seconds; that is, the maximum time to wait for missed packets before performing the trigger calculations. ◆ Permitted values: Any integer from 0 to 300. Example: `CompletionDelay = 10`
HighPassOrder	The filter order of the highpass filter for this trigger. ◆ Permitted values: 0 to 5, with the restriction that HighPassOrder + LowPassOrder <= 5. Example: `HighPassOrder = 2`
HighPassCornerHz	The 3dB corner frequency in hertz of the highpass filter for this trigger. ◆ Permitted values: Any value from 0.0 to 999.0. Example: `HighPassCornerHz = 1.0`
LowPassOrder	The filter order of the lowpass filter for this trigger. ◆ Permitted values: 0 to 5, with the restriction that HighPassOrder + LowPassOrder <= 5. Example: `LowPassOrder = 3`
LowPassCornerHz	The 3dB corner frequency in hertz of the lowpass filter for this trigger. ◆ Permitted values: Any value from 0.0 to 999.0. Example: `LowPassCornerHz = 8.0`
StaConstant	The default STA (short-term average) time constant in seconds for this trigger. ◆ Permitted values: Any value from 0.01 to 999.0. Example: `StaConstant = 0.1`
LtaConstant	The default LTA (long-term average) time constant in seconds for this trigger. ◆ Permitted values: Any value from 0.01 to 999.0. Example: `LtaConstant = 20`
TriggerRatio	The STA/LTA ratio at which the trigger becomes active. ◆ Permitted values: Any positive integer. Example: `TriggerRatio = 2`
EarlyReportDelay	The delay time in seconds between the time at which the trigger level is first exceeded and the time of the early trigger report. This delay is used to obtain an early estimate of the signal amplitude and frequency of the triggering signal. ◆ Permitted values: Any value from 0.1 to 5.0. Example: `EarlyReportDelay = 1.0`

2.3.11 [Trigger]

This section defines a channel-specific trigger detector. A channel-specific trigger is obtained by applying a named detector type to a specific data channel; the configuration must specify the type and the channel name. It contains the parameters listed in Table 2-21 and may contain the optional parameter listed in Table 2-22.

- Create a separate section for each trigger.

Table 2-21 [Trigger] Section Parameters

Parameter	Definition
Type	The name of the named detector type to use for this trigger. ◆ Permitted values: Any previously-defined detector type *Name*. Example: `Type = Teleseism`
Channel	The dotted station-channel name of the time series for this trigger. Only previously-defined station-channel names will produce active triggers. ◆ Permitted values: Any string. All letters in the name must be uppercase. Example: `Channel = STN05.BHZ`

Table 2-22 Optional Parameter for [Trigger]

Parameter	Definition
TriggerRatio	The STA/LTA ratio at which the trigger becomes active. ◆ Permitted values: Any positive integer. Example: `TriggerRatio = 2`

Chapter 3
Running NAQSServer

This chapter provides information on stopping and starting NAQSServer, using the run-time commands, and monitoring the operation of NAQSServer.

3.1 Starting and Stopping NAQSServer

3.1.1 Starting NAQSServer Locally

In a typical network, NAQSServer is set up to start automatically using scripts (on Linux and Solaris) or the NmxWatchdog program (on Windows). It can also be started manually from a command line.

- To start NAQSServer manually on a Windows computer:
  - Type the following at the command prompt: `C:\>/nmx/user/NAQSServer`
- To start NAQSServer manually on a Solaris or Linux computer:
  - Type the following command into any terminal window: `naqs start`

3.1.2 Running NAQSServer with NmxWatchdog on Windows

NAQSServer can be started automatically and monitored by the Nanometrics watchdog program on a Windows computer.

- To run NAQSServer with NmxWatchdog add the following entry to the `watchdog.ini` file:

```
[ WatchEntry n ]
  ProgramTitle = "NaqsServer"
  ProgramPathname = "java -Xrs -ms5m -cp c:\nmx\bin\NaqsServer.jar
ca.nanometrics.naqsserver.NaqsServer"
  WorkingDirectory = "c:\nmx\user"
  ExitAction = Restart
  PingsSemaphore = TRUE
  StartDelay = 6s
```

3.1.3 Stopping NAQSServer Locally

NAQSServer must be shut down properly to release its system resources.

- To stop NAQSServer locally on a Windows computer:
  - Type the following command into the NAQSServer terminal window: `quit`
- To stop NAQSServer locally on a Solaris or Linux computer:
  - Type the following command into any terminal window: `naqs stop`

3.1.4 Stopping NAQSServer Remotely

- To stop NAQSServer on a remote Windows computer, type the following command into a telnet session: `Nmxkill c:\nmx\user\NaqsServer`
- To stop NAQSServer on a remote Solaris or Linux computer:
  Type the following command into any terminal window: `naqs stop`

3.2 Using the Run-Time Commands

NAQSServer supports a basic keyboard interface for entering run-time commands, with the options described in Table 3-1.

- To enter runtime commands in the NAQSServer terminal window, enter *command*.

Table 3-1 NAQSServer Run-Time Commands

To do this...	Enter this command...
Display all log messages in the log file; set the log verbosity to DEBUG	`D`
Suppress debug messages in the log file; set the log verbosity to VERBOSE	`V`
Suppress debug and verbose messages in the log file; set the log verbosity to INFO	`I`
Move the log file (close the current log and start a new file)	`M`
Stop NAQSServer and exit: on Windows on Solaris and Linux	`quit` `stop`
Request summary status information written to the log.* Options, with the corresponding keyword listed at right, are:	`REPORT` *keyword keyword2 keyword3 ...*
the amount of time since NAQSServer was started or restarted	`UPTIME`
the number of packets received by NAQSServer	`RX`
the number of packets sent by NAQSServer	`TX`
the number of triggers detected by each detector since midnight or startup	`TRIGGERS`
the number of events detected since midnight or startup	`EVENTS`
packet statistics for each channel	`CHANNELS`
the number of new datastream connections since midnight, and the current connections	`CONNECTIONS`

Table 3-1 NAQSServer Run-Time Commands (Continued)

To do this...	Enter this command...
information about each data subscriber (address/port and number of channels subscribed)	`SUBSCRIBERS`
all of the above	`ALL`
Monitor packet statistics for selected channels at user specified intervals.*	`MONITOR` *channelname interval*
Do not request missed packets	`RETX OFF`
Request missed packets	`RETX ON`

*The `REPORT` and `MONITOR` commands report the following packet statistics:

P – The number of packets received since the last reset.
M – The number of packets missed since the last reset.
R – The number of missed packets recovered since the last reset.
C – The number of packets (and contiguous gaps) currently missing.
U – The number of missing packets (and contiguous gaps) that cannot be recovered.
T – The number of seconds since the last reset.
L – The number of seconds since receiving data.

3.3 Monitoring the Operation of NAQSServer

3.3.1 Using the NAQSServer Log

NAQSServer generates log messages that trace the operation of the program. It displays these messages in the terminal window and writes them to the NAQSServer log file. You can set the level of detail (the verbosity) of the information to be displayed and recorded.

- To view the log, open the log file *LogFile_date.log* in a text editor. The log file name and location are set in the [NaqsLog] section of the `Naqs.ini` configuration file.
- To set the verbosity of log messages on startup, edit the [NaqsLog] section of the `Naqs.ini` configuration file.
- To change the verbosity of log messages while NAQSServer is running, use the runtime commands (Section 3.2).
- To generate a report from the log, use the `REPORT` runtime command (see Table 3-1 on page 34).

3.3.2 Interpreting NAQSServer Alert Messages

NAQSServer issues messages alerting the network manager or other interested parties of the acquisition system status or significant events which have occurred. These messages may be reformatted and forwarded via email using the AlertMailer program. Each message is assigned a priority which reflects its severity. (See also the AlertMailer user guide and the Nanometrics Data Formats user guide.)

Note The system time of the NAQSServer computer must be synchronized with the Universal Time Coordinated (UTC) standard.

NAQSServer issues the alert messages described in Table 3-2.

Table 3-2 NAQSServer Alert Messages

Alert message	Description	Priority
NaqsAlive	◆ Issued on startup ◆ Gives a list of which instruments are online and which instruments are offline	1
NaqsStatus	◆ Issued once per hour ◆ Gives a list of which instruments are online and which instruments are offline	1
NaqsReport	◆ Issued once per day ◆ Gives uptime, packet count, and a list of instruments which were online or offline for the past 24 hours	1
RbfOpenFail	◆ Issued if one or more ringbuffers cannot be opened properly ◆ This condition usually indicates lack of disk space	The number of ringbuffers which were not opened
RbfWriteFail	◆ Issued when there is an error writing to a ringbuffer (after 1, 100, 1000, or *N* * 10000 write errors) ◆ This condition usually will require user intervention	The number of write errors on this channel
RbfWriteOk	◆ Issued when a ringbuffer write succeeds after previously being failed	The same as that in the corresponding RbfWriteFail message
InstrumentOffline	◆ Issued when Naqs has not received any data for 10 minutes from one or more instruments that were previously online ◆ Gives a list of instruments that just went offline, plus a status summary for all instruments	10
InstrumentOnline	◆ issued when Naqs starts receiving from one or more instruments that were previously offline ◆ Gives a list of instruments that just came online, plus a status summary for all instruments	10
NaqsEvent	◆ Issued when the event-detection module detects a seismic event	The average over all triggers of the peak STA/LTA ratio recorded on each trigger

Appendix A
Configuration File Examples

A.1 Naqs.ini File Example

```
// Naqs.ini
// Simple ini-format configuration file for NaqsServer

[ NaqsLog ]
LogPath = /nmx/logs/Naqs       // directory in which to store the log file
LogFile = Naqs.log             // base name for the log file
Verbosity = INFO               // start-up verbosity (normally use INFO)

[ AlertSender ]
InetHost = none        // host or multicast address for alert messages
Port = 31000           // port to which to send alert messages
SourceId = Naqs-A      // name by which alerts from this program will be identified

[ NetworkInterface ]
Port = 32000                 // UDP port for incoming NMX data (usually 32000)
SendDelay = 250              // milliseconds to delay after each send
RetxRequest = Enabled
MulticastGroup = 224.1.1.1

[ EventAssociator ]
EventPath = /nmx/events      // directory in which to store the event file
EventFile = Naqs.elf         // base name for the event file
TrigsToStart = 2             // number of triggers to declare an event
CoincidenceWindow = 20       // coincidence window in seconds
EventTimeout = 60            // maximum duration of event in seconds
SaveTriggers = Yes           // write triggers to ELF file (Yes/No)

[ Datastream ]
Port = 28000                 // TCP port for control/data connections to Naqs
Password = none              // access password
MaxConnections = 10          // maximum number of simultaneous connections
SocketType = Direct          // connection type (Direct or Callback)
DataBufferLength = 30        // Buffer length for data channels (# packets)

[ Calibrator ]
Password = calpass// Password for remote calibration
```

A.2 Naqs.stn File Example

```
// ---------------------------------------------------------------------------------
// Naqs.stn Station database file for NaqsServer
// ---------------------------------------------------------------------------------

// ---------------------------------------------------------------------------------
// This section defines the network name and acquisition system ID.
// ---------------------------------------------------------------------------------

[ Network ]
Name = NMX
NaqsID = A


// ---------------------------------------------------------------------------------
// The following sections define the sensors used in this network.
// ---------------------------------------------------------------------------------

[ Sensor ]                  // predefined sensor - all fields mandatory
TypeName = TRILLIUM120P     // name of this prototype - may be same as model
Model = TRILLIUM120P        // sensor model name
SensitivityUnits = M/S      // units of ground motion:  M, M/S or M/S**2
Sensitivity = 0.48e+9        // counts per unit of ground motion (System Sensitivity)
SensitivityFreq = 1.0       // Frequency at which sensitivity is correct
CalibrationUnits = VOLTS    // calibration input units:  VOLTS or AMPS
CalCoilResistance = 9200    // calibration coil resistance in ohms
CalCoilConstant = 100       // Calibration units per m/s/s
CalEnable = -1              // digital enable signal for calibration
CalRelay  = 0               // analog relay for calibration (0 = use channel number)
MassCenterEnable = -1       // digital enable signal for mass centering
MassCenterDuration = 5     // duration of mass centering signal in seconds (optional)
CalSource = Trident         // gives the source of the Cal signal

[ Sensor ]                  // predefined sensor - all fields mandatory
TypeName = Wind Sensor      // name of this prototype - may be same as model
Model = Wind Sensor         // sensor model name
SensitivityUnits = M/S      // units of ground motion:  M, M/S or M/S**2
Sensitivity = 1.0e+9        // counts per unit of ground motion (System Sensitivity)
SensitivityFreq = 1.0       // Frequency at which sensitivity is correct
CalibrationUnits = VOLTS    // calibration input units:  VOLTS or AMPS
CalCoilResistance = 20000   // calibration coil resistance in ohms
CalCoilConstant = 33.2      // Calibration units per m/s/s
CalEnable = 2               // digital enable signal for calibration
CalRelay  = 0               // analog relay for calibration (0 = use channel number)
MassCenterEnable = 1        // digital enable signal for mass centering
MassCenterDuration = 5     // duration of mass centering signal in seconds (optional)
CalSource = Trident         // gives the source of the Cal signal


// ---------------------------------------------------------------------------------
// The following sections define the instruments used in this network.
// ---------------------------------------------------------------------------------

[ InstrumentPrototype ]     // predefined instrument - all fields mandatory
TypeName = CARINA           // prototype name - may be same as model
Model = CARINA              // instrument type
MemoryKB  = 12000           // instrument ReTx buffer size
SohBundlesPerPacket = 19    // bundles per soh packet
RequestInterval = 300       // interval between retx request messages
SohChannelName = CAR        // extension for soh file (NUL if none)
SohBufferSize = 50          // file size in MB
SohBufferPath = /nmx/soh  // where files are located
```

```
InetHostName = Dynamic      // return IP address for instrument
InetPort = 32000            // return IP port for instrument

[ InstrumentPrototype ]     // predefined instrument - all fields mandatory
TypeName = CYGNUS           // prototype name - may be same as model
Model = CYGNUS              // instrument type
MemoryKB  = 12000           // instrument ReTx buffer size
SohBundlesPerPacket = 19    // bundles per soh packet
RequestInterval = 300       // interval between retx request messages
SohChannelName = CYG        // extension for soh file (NUL if none)
SohBufferSize = 50          // file size in MB
SohBufferPath = /nmx/soh  // where files are located
InetHostName = Dynamic      // return IP address for instrument
InetPort = 32000            // return IP port for instrument

[ InstrumentPrototype ]     // predefined instrument - all fields mandatory
TypeName = TRIDENT          // prototype name - may be same as model
Model = TRIDENT             // instrument type
MemoryKB  = 3200            // instrument ReTx buffer size
SohBundlesPerPacket = 15    // bundles per soh packet
RequestInterval = 300       // interval between retx request messages
SohChannelName = TRI        // extension for soh file (NUL if none)
SohBufferSize = 50          // file size in MB
SohBufferPath = /nmx/soh  // where files are located
InetHostName = Dynamic      // return IP address for instrument
InetPort = 32000            // return IP port for instrument


// ---------------------------------------------------------------------------
// The following sections define the prototype channels used in this network.
// ---------------------------------------------------------------------------

[ ChannelPrototype ]        // predefined channel - all fields mandatory
TypeName = BHZ-1            // label for this type
Name = BHZ                  // channel name
Component = 1               // digitizer component (refers to current instrument)
Sensor = TRILLIUM120P       // pointer to predefined [Sensor] characteristics
Azimuth = 0                 // azimuth in degrees clockwise from North
Dip = 90                    // dip in degrees (positive down)
Depth = 0                   // has to be defined for each channel
BundlesPerPacket = 19       // number of bundles per data packet
RingBufferSize = 200        // file size in MB
RingBufferPath = /nmx/ringbuffer           // where files are located
ResponseFile = /nmx/user/seed_TRL120P.rsp  // name of SEED response file

[ ChannelPrototype ]        // predefined channel - all fields mandatory
TypeName = BHN-1            // label for this type
Name = BHN                  // channel name
Component = 2               // digitizer component (refers to current instrument)
Sensor = TRILLIUM120P       // pointer to predefined [Sensor] characteristics
Azimuth = 0                 // azimuth in degrees clockwise from North
Dip = 0                     // dip in degrees (positive down)
Depth = 0                   // has to be defined for each channel
BundlesPerPacket = 19       // number of bundles per data packet
RingBufferSize = 200        // file size in MB
RingBufferPath = /nmx/ringbuffer           // where files are located
ResponseFile = /nmx/user/seed_TRL120P.rsp  // name of SEED response file

[ ChannelPrototype ]        // predefined channel - all fields mandatory
TypeName = BHE-1            // label for this type
Name = BHE                  // channel name
Component = 3               // digitizer component (refers to current instrument)
Sensor = TRILLIUM120P       // pointer to predefined [Sensor] characteristics
```

```
Azimuth = 90                // azimuth in degrees clockwise from North
Dip = 0                     // dip in degrees (positive down)
Depth = 0                   // has to be defined for each channel
BundlesPerPacket = 19       // number of bundles per data packet
RingBufferSize = 200        // file size in MB
RingBufferPath = /nmx/ringbuffer           // where files are located
ResponseFile = /nmx/user/seed_TRL120P.rsp  // name of SEED response file

[ ChannelPrototype ]        // predefined channel - all fields mandatory
TypeName = BWS              // label for this type
Name = BWS                  // channel name
Component = 1               // digitizer component (refers to current instrument)
Sensor = Wind Sensor        // pointer to predefined [Sensor] characteristics
Azimuth = 0                 // azimuth in degrees clockwise from North
Dip = 0                     // dip in degrees (positive down)
Depth = 0                   // has to be defined for each channel
BundlesPerPacket = 19       // number of bundles per data packet
RingBufferSize = 200        // file size in MB
RingBufferPath = /nmx/ringbuffer   // where files are located
ResponseFile = none         // name of SEED response file

[ ChannelPrototype ]        // predefined channel - all fields mandatory
TypeName = BWD              // label for this type
Name = BWD                  // channel name
Component = 2               // digitizer component (refers to current instrument)
Sensor = Wind Sensor        // pointer to predefined [Sensor] characteristics
Azimuth = 0                 // azimuth in degrees clockwise from North
Dip = 0                     // dip in degrees (positive down)
Depth = 0                   // has to be defined for each channel
BundlesPerPacket = 19       // number of bundles per data packet
RingBufferSize = 200        // file size in MB
RingBufferPath = /nmx/ringbuffer   // where files are located
ResponseFile = none         // name of SEED response file



// ---------------------------------------------------------------------------------
// The following sections define the stations used in this network and their
// associated instruments and channels
// ---------------------------------------------------------------------------------

// ---------------------------------------------------------------------------------
// Central Recording Facility
// ---------------------------------------------------------------------------------

[ Station ]
Name = CRF
Description = Central Hub
Latitude = -6.1547
Longitude = 106.8416
Elevation = 40

[ Instrument ]              // instance of an instrument
Prototype = CARINA          // instrument type
SerialNumber = 123          // serial number - mandatory
```

```
// ---------------------------------------------------------------------------------
// Remote Stations
// ---------------------------------------------------------------------------------

// ---------------------------------------------------------------------------------
[ Station ]
Name = STN01
Description = Remote Station 1: Trillium 120P
Latitude = -3.700
Longitude = 128.0833
Elevation = 100

[ Instrument ]              // instance of an instrument
Prototype = CYGNUS          // instrument type
SerialNumber = 650          // serial number - mandatory

[ Instrument ]              // instance of an instrument
Prototype = TRIDENT         // instrument type
SerialNumber = 932          // serial number - mandatory

[ Channel ]                 // instance of a channel
Prototype = BHZ-1           // use settings from this prototype
[ Channel ]                 // instance of a channel
Prototype = BHN-1           // use settings from this prototype
[ Channel ]                 // instance of a channel
Prototype = BHE-1           // use settings from this prototype


// ---------------------------------------------------------------------------------
[ Station ]
Name = STN02
Description = Remote Station 2: Trillium 120P and Wind Sensor
Latitude = -6.42
Longitude = 106.85
Elevation = 100

[ Instrument ]              // instance of an instrument
Prototype = CYGNUS          // instrument type
SerialNumber = 659          // serial number - mandatory

[ Instrument ]              // instance of an instrument
Prototype = TRIDENT         // instrument type
SerialNumber = 909          // serial number - mandatory

[ Channel ]                 // instance of a channel
Prototype = BHZ-1           // use settings from this prototype
[ Channel ]                 // instance of a channel
Prototype = BHN-1           // use settings from this prototype
[ Channel ]                 // instance of a channel
Prototype = BHE-1           // use settings from this prototype

[ Instrument ]              // instance of an instrument
Prototype = TRIDENT         // instrument type
SerialNumber = 1035         // serial number - mandatory

[ Channel ]                 // instance of a channel
Prototype = BWS             // use settings from this prototype
[ Channel ]                 // instance of a channel
Prototype = BWD             // use settings from this prototype
```

```
// -------------------------------------------------------------------------------
// This section shows how to specify detectors and triggers.
// -------------------------------------------------------------------------------

[ DetectorType ]
Name = Teleseism
TypeID = 1
CompletionDelay = 10
HighPassOrder = 2
HighPassCornerHz = 1.0
LowPassOrder = 3
LowPassCornerHz = 8.0
StaConstant = 0.1
LtaConstant = 20
TriggerRatio = 6
EarlyReportDelay = 1.0

[ Trigger ]
Type = Teleseism
Channel = STN01.BHZ
```

About Nanometrics

Nanometrics leads the world in the development of digital technology and networks for seismological and environmental studies. The award-winning Canadian exporter was the first company to produce a fully-integrated satellite system specially designed for studying and monitoring earthquakes.

Nanometrics has customers on every continent in more than 200 different countries. Our customers have used our technology to establish and grow research networks across every environment in the world from the frozen tundra of Canada's north to the arid deserts of the Middle East to the jungles of South America. Many of these include mission-critical national and regional networks that demand the highest possible data quality and availability.

Contacting Nanometrics

Nanometrics Inc.
250 Herzberg Road
Kanata, Ontario, Canada K2K 2A1
Phone: +1 613-592-6776
Fax: +1 613-592-5929
Email: info@nanometrics.ca
Web: www.nanometrics.ca

Contacting Customer Support

If you need technical support please submit a request on the Nanometrics customer support site or by email or fax. Include a full explanation of the problem and related information such as log files.

Support site:	http://support.nanometrics.ca
Email:	techsupport@nanometrics.ca