

Q330 Communications Library Interface Specification

This document contains information that is proprietary to Quanterra, Inc. Use of the information in this document other than an informational description of a Quanterra product in development is strictly prohibited. Photocopying or electronic transmission of this document without the prior written permission of Quanterra, Inc. is prohibited.

Changes in revision five are marked with a **5** Rev six marked with a **6** Rev seven marked with a **7** Rev eight marked with a **8**. Rev 10 reverses the changes in Rev 9. Changes in revision eleven are marked with a **11**


Some features are only available when connected to a Q330, these are marked with a **Q330**. Any features only available when connected to a Q335 are marked with a **Q335**

Library Operation Overview

1. Create the station thread context. This starts a thread for the station and returns an opaque pointer to the “station context”. If it has already been created from a previous invocation then the “thread continuity” file will also be read. From the contents of this file DP statistics will be restored and if configured in the DP Tokens DP data streams will start to be generated. Regardless of whether a thread continuity file is found or not a LOG DP LCQ will be created to hold messages⁴. At this point the library is in the Idle state. If archival miniseed is configured then the library will also go through each of the DP data tokens and request the last Seed record from the host in order to append data to the last record rather than starting a new record.
2. Start Registration. If the Q330 does not have a dynamic IP address then the library attempts to contact the Q330 else it will go into a “Wait” state until a POC is received. Once registration is complete and tokens have been read then the library will read the “Q330 continuity” file to restore internal data structures (such as decimation filters and detectors) to the state they were in when last disconnecting from the Q330. If archival miniseed is configured then the library will also go through each of the Q330 data tokens and request the last Seed record from the host in order to append data to the last record rather than starting a new record. The library waits in the “Run Wait” state before accepting data from the Q330. This state is used by the host to start any netserver or a webserver based on the TCP port numbers contained in the tokens. Note that while Seneca waits until this point to start a netserver (if configured) Pecos2/3 actually starts these up after the first step based on stored values and keeps them running until the last step. Once the host has done any steps it requires then it requests the library enter the “Run” state. The library automatically switches between states based on the status of communications with the Q330.
3. Stop Registration. The host requests that the library change to the Idle state. If the library is currently connected to the Q330 it goes through the steps to required to disconnect. Anytime the library disconnects from the Q330 it will flush any Seed buffers and writes the “Q330 continuity” file, this includes disconnections from communications loss.
4. Request thread termination. The library will flush any miniseed records for the DP data to allow the host to save them and then terminate the station thread.
5. Destroy the station thread context. Does the final cleanup.

Conventions

The library code is translated from Pascal to C. An include file called pascal.h (in conjunction with platform.h) is used to allow the minimum number of changes between versions. As a result you might not recognize the type specifications used, they are:

- Byte = unsigned byte
- Shortint = signed byte
- Boolean = unsigned byte (0 = False, 1 = True)
- Word = unsigned 16 bits
- Int16 = signed 16 bits
- Longword = unsigned 32 bits
- Longint = signed 32 bits
- Integer = native integer size of the host platform (C “int”), must be at least 32 bits. Must be same size as a pointer on the host platform. 
- Pointer = pointer to anything (C “void *pointer”)
- Single = 32 bit IEEE floating point
- Double = 64 bit IEEE floating point
- String = null terminated character array of 256 bytes
- Stringxx = null terminated character array of xx + 1 bytes

Primary File Descriptions

In general entry points that start with “lib_XXXX” are intended for host programs. There are other entry points defined but these often require the use of the internal context pointer which the host should not be using (except as an opaque pointer for the lib_XXXX entry points).

- Libtypes – Contains the constants and types that a host program is likely to require to interpret data from the Q330 such as configuration and status packets. Also contains the definitions for the library error codes and states.
- Libclient – Constants, types, and entry points that a host would use to communicate with the library.
- Libmsgs – Contains verbosity bit values and all of the message codes. Also contains routines to translate message codes, error codes, and other enumerated types into strings.
- Platform – Contains mapping of C data types and global includes based on Architecture.

Secondary File Descriptions

These files contain constants, types, and routines that may be of use to a host program.

- Libseed – Definitions for Seed headers and records and routines to store and load various parts of those records from the host native format to valid Seed records and from Seed records to host native format.
- Libcvrt – Basic store and load routines for converting between host native format and big-endian communications/Seed formats.
- Libnetserv – Creates a thread and socket to handle one netserv (LISS) connection. Multiple netserves may be run. The host program is responsible for transferring any data from the library 512byte miniseed callback to the netserver(s).
- Libpoc – Creates a thread and socket to handle receiving SeisPOC messages. The host can then obtain these via a callback and dispatch them to the appropriate station thread.
- Libsupport – Contains some utility routines that may be of use.

Tertiary File Descriptions

A host program is unlikely to require these files.

- Libcmds – This file contains the command handling for the communications with the Q330.
- Libstrucs – The internal data structures for the station thread
- Libmd5 – Authentication code to register with a Q330.
- Libsampglob – Data structures for handling data from the Q330.
- Libsampcfg – Setup routines dealing with the above structures.
- Libsample – Conversion of data from the Q330 into miniseed
- Libslider – Sliding window handling for the data packets from the Q330
- Libcont – Handles both thread and Q330 continuity files.
- Libcompress – Steim2 compression routines for generating miniseed.
- Libopaque – Handles generating opaque blockettes in the miniseed from configuration data
- Libdetect – Threshold and Murdock-Hutt detectors
- Libctrldet – Control detector routines
- Liblogs – Miniseed message log generation.
- Libarchive – Handles archival miniseed (typically 4K records)
- Libstats – Produces DP statistics and as well as the DP statistics data streams if enabled.
- Libtokens – Handles decoding of tokens from the Q330 into internal data structures.
- Libdss – Data Subscription Service handler. **7**
- Q330io – Socket and serial port routines for communications with the Q330.
- Q330types – Constants and data structures for communicating with the Q330.
- Q330cvrt – Routines to convert between host native format and Q330 data packets.

Routines in Libclient

```
extern void lib_create_context (tcontext *ct, tpar_create *cfg) ;
```

Setup “cfg” with the required parameters. If the thread was created then the variable pointed to by “ct” will have the opaque “station context” pointer. If not then cfg.resp_err will have an error code. Tpar_create is defined as:

```
typedef struct { /* parameters for lib_create call */
    t64 q330id_serial ; /* serial number */
    word q330id_dataport ; /* Data port, use LP_TEL1 .. LP_TEL4 */
    string5 q330id_station ; /* initial station name */
    longint host_timezone ; /* seconds offset of host time. Initial value if auto-adjust enabled */
    string95 host_software ; /* host software type and version */
    string250 opt_contfile ; /* continuity root file path and name, null for no continuity */
    word opt_verbose ; /* VERB_xxxx bitmap */
    word opt_zoneadjust ; /* calculate host's timezone automatically */
    word opt_secfilter ; /* OSF_xxx bits */
    word opt_client_msgs ; /* Number of client message buffers */
#ifdef OMIT_SEED
    word opt_compat ; /* Compatibility Mode */ 5
    word opt_minifilter ; /* OMF_xxx bits */
    word opt_aminifilter ; /* OMF_xxx bits */
    word amini_exponent ; /* 2**exp size of archival miniseed, range of 9 to 14 */
    integer amini_512highest ; /* rates up to this value are updated every 512 bytes */
    word mini_embed ; /* 1 = embed calibration and event blockettes into data */
    word mini_separate ; /* 1 = generate separate calibration and event records */
    pfilter mini_firchain ; /* FIR filter chain for decimation */
    tcallback call_minidata ; /* address of miniseed data callback procedure */
    tcallback call_aminidata ; /* address of archival miniseed data callback procedure */
#endif
    enum tliberr resp_err ; /* non-zero for error code creating context */
}
```

```

tcallback call_state ; /* address of state change callback procedure */
tcallback call_messages ; /* address of messages callback procedure */
tcallback call_secdata ; /* address of one second data callback procedure */
tcallback call_lowlatency ; /* address of low latency data callback procedure */
tcallback call_baler ; /* Baler related callbacks */ 4
pfile_owner file_owner ; /* For continuity file handling */ 4
} tpar_create ;

```

- t64 q330id_serial – Set to the 8 byte serial number of the Q330 you wish to connect to. Note that the library does not assume your system has a 64 bit integer type, all large integer types are made up of an array of smaller type. In the case of “t64” this is an array of two 32 bit unsigned values. Each “longword” is in host byte endian and for big-endian hosts the most significant longword is array element 0 while on little-endian hosts the most significant longword is array element 1.
- word q330id_dataport – A value of zero (for data port 1) to three (for data port 4)
- string5 q330id_station – Used until actual station name is read from tokens (or thread continuity)
- longint host_timezone – If your computer is on UTC then this would be zero, else the number of seconds to be added to your computer’s clock to get UTC. See comments below regarding opt_zoneadjust.
- string95 host_software – Used to identify the host software in the message log if enabled in verbosity.
- string250 opt_contfile – Path name to root continuity file name. Null string to disable continuity. The library will append a “t” to the path for the thread continuity and a “q” for Q330 continuity.
- word opt_verbose – Any combination (by oring) of the following:
 - VERB_SDUMP – Enables a dump of Q330 status to the message log when connecting.
 - VERB_RETRY – Logs command retries.
 - VERB_REGMSG – Generates a “Ping” to the Q330 and a user message when registering and de-registering.
 - VERB_LOGEXTRA – Adds messages for items like filter delays.
 - VERB_AUXMSG – Enables 800 series messages (from netserver and webserver)
 - VERB_PACKET – Logs packets sent to and received from the Q330.
- word opt_zoneadjust – If set non-zero then the library will automatically calculate the timezone (30 minute increments) between UTC from the Q330 and the host computer. This information will be stored in the thread continuity so it will be automatically restored. If zero then the “host_timezone” value above will be used.
- word opt_secfilter – Determines which data will be sent via the one second callback assuming there is a callback specified. Note that you can specify more than one option by “oring” them together. For instance if you set this field to OSF_ALL or OSF_DATASERV then you will get all packets, but you can tell (if you care) which ones were due to Dataserv by looking at the “filter_bits” field in the callback structure.
 - OSF_ALL – Sends all one second data.
 - OSF_DATASERV – Sends data that have “Dataserv” enabled in their token.
 - OSF_1HZ – Sends main digitizer 1Hz data (3 or 6 channels).
 - OSF_EP – Sends Environmental Processor 1Hz data. 6
- word opt_client_msgs – Number of buffers to allocate for messages generated by the client program (using lib_msg_add) or by other libraries. Minimum is 10 if not set higher.

```
#ifndef OMIT_SEED
```

- word opt_minifilter – Similar to the one second callback this is used to filter the miniseed (512 byte) data. Likewise you can tell what criteria was used by looking at the filter_bits field.
 - OMF_ALL – Send all.
 - OMF_NETSERV – Sends data that have “Netserv” enabled in their token.
 - OMF_CFG – Send configuration opaque blockette based records.
 - OMF_TIM – Timing log records.
 - OMF_MSG – Message log records.
- word opt_aminifilter – Same as “opt_minifilter” above except for the archival miniseed.
- Word opt_compat – If non-zero then 512 byte output will be set to event-only if archival output is event-only. If zero then 512 byte output will be controlled by recently added flag bit in token. 5
- word amini_exponent – The “exponent” for the miniseed record for archival miniseed generation, for example, 9 = 512 byte records (minimum possible), 12 = 4096 byte records, and 14 = 16384 byte records (maximum possible).
- integer amini_512highest – Indicates the highest data rate where the archival miniseed uses “incremental” updates (described in the callback structure area later). For instance 20 would be 20Hz, -10 would be 0.1Hz.
- word mini_embed – Set non-zero to embed calibration and event blockettes into the miniseed data records they are associated with.

- word mini_separate – Set non-zero to generate separate miniseed records for each calibration or event detection blockette. This would be required for Comserv compatibility.
- pfilter mini_firchain – If set non-NIL this is the start of a FIR filter chain to be added to the library's built in chain. The library includes the decimate by 10 filters DEC10, VLP389, and ULP379.
- tcallback call_minidata – The address of the 512 byte miniseed callback routine if non-NIL.
- tcallback call_aminidata – The address of the archival miniseed callback routine if non-NIL.

#endif

- enum tliberr resp_err – Returns the error code for creating context, LIBERR_NOERR if none.
- tcallback call_state – The address of the state callback routine (see description of this callback later)
- tcallback call_messages – The address of the message callback routine (see description of this callback later)
- tcallback call_secdata – The address of the one second callback routine (see description of this callback later)
- tcallback call_lowlatency – The address of the low-latency callback routine, not currently implemented.
- tcallback call_baler – The address of the baler related callback routine (see description of this callback later)
- pfile_owner file_owner – The address of the file handling structure (see description of the structure later)

```
extern enum tliberr lib_register (tcontext ct, tpar_register *rpar) ;
```

Setup "rpar" with the registration parameters. "ct" is the pointer returned by lib_create_context. If there is domain name or IP address specified then the library will attempt to open a connection with the Q330, else it will wait for a POC. "tpar_register" is defined as:

```
typedef struct { /* parameters for lib_register call */
    t64 q330id_auth ; /* authentication code */
    string250 q330id_address ; /* domain name or IP address in dotted decimal */
    word q330id_baseport ; /* base UDP port number */
    enum thost_mode host_mode ;
    string250 host_interface ; /* ethernet or serial port path name */
    word host_mincmdretry ; /* minimum command retry timeout */
    word host_maxcmdretry ; /* maximum command retry timeout */
    word host_ctrlport ; /* set non-zero to use specified UDP port at host end */
    word host_dataport ; /* set non-zero to use specified UDP port at host end */
#ifdef OMIT_SERIAL
    word serial_flow ; /* 1 = hardware flow control */
    longword serial_baud ; /* in bps */
    longword serial_hostip ; /* IP address to identify host */
#endif
    word opt_latencytarget ; /* seconds latency target for low-latency data */
    word opt_closedloop ; /* 1 = enable closed loop acknowledge */
    word opt_dynamic_ip ; /* 1 = dynamic IP address */
    word opt_hibertime ; /* hibernate time in minutes if non-zero */
    word opt_conntime ; /* maximum connection time in minutes if non-zero */
    word opt_connwait ; /* wait this many minutes after connection time or buflevel shutdown */
    word opt_regattempts ; /* maximum registration attempts before hibernate if non-zero */
    word opt_ipexpire ; /* dyanmic IP address expires after this many minutes since last POC */
    word opt_buflevel ; /* terminate connection when buffer level reaches this value if non-zero */
    word opt_q330_cont ; /* Determines how often Q330 continuity is written to disk in minutes */
    word opt_dss_memory ; /* Maximum DSS memory (in KB) if non-zero */
} tpar_register ;
```

- t64 q330id_auth – Set to the authentication code (probably zero) for that Q330 and the required data port.
- string250 q330id_address – This can either be a dotted-decimal IPV4 address or a domain name for the library to look up. A value of "255.255.255.255" puts the library into "Baler Mode" and "q330id_auth" must have the special baler authentication code.
- word q330id_baseport – Q330 Base Port, normally 5330.
- enum thost_mode host_mode – This is "HOST_ETH" for Ethernet, "HOST_SER" for serial, or "HOST_TCP" for TCP using Tunnel330.⁶
- string250 host_interface – Normally a Null string for Ethernet. For serial it might be something like "COM2" for a windows system or "/dev/ttyb" for a Unix based system.
- word host_mincmdretry – Minimum command retry time in 100ms increments. This clips the library calculated value so it does not go below this value.
- word host_maxcmdretry – Maximum command retry time in 100ms increments. This clips the library calculated values so it does not go above this value.
- word host_ctrlport – Set to zero to use the normal OS generated host port for the command connection to the Q330, else set to the port you desire.

- word host_dataport – Set to zero to use the normal OS generated host port for the data connection to the Q330, else set to the port you desire.
- ```
#ifndef OMIT_SERIAL
```
- word serial\_flow – Enables hardware flow control for serial connections.
  - longword serial\_baud – Sets the baud for the serial connection.
  - longword serial\_hostip – Sets the IP address that the host will use for it's end of a serial connection.
- ```
#endif
```
- word opt_latencytarget – Not currently used.
 - word opt_closedloop – Not currently used.
 - word opt_dynamic_ip – Set non-zero if the Q330 has a dynamic IP address. If you don't know what the IP address of the Q330 is when calling this routine then set "q330id_address" to a Null string so the library won't attempt to start a connection until a POC is received.
 - word opt_hibertime – If "opt_regattempts" (below) is non-zero and the library tries and fails that many times to register with the Q330 then it wait this many minutes before starting a new cycle of registration attempts.
 - word opt_conntime – If this is non-zero and a connection is established for this many minutes then the library will automatically de-register with the Q330 and then wait "opt_connwait" (below) minutes before trying to register again.
 - word opt_connwait – If either the "opt_conntime" or "opt_buflevel" values are reached then the library will wait this many minutes before trying to register again.
 - word opt_regattempts – If non-zero and the library tries and fails to register with the Q330 this many times it will stop trying until the "opt_hibertime" timeout expires.
 - word opt_ipexpire – If non-zero then this limits the "lifetime" of a dynamic IP address received via a POC to this many minutes.
 - word opt_buflevel – If non-zero then disconnect from the Q330 once the packet buffer memory has been reduced to the indicated percent.
 - word opt_q330_cont – If non-zero then limits how often the Q330 continuity file will be written to disk (in minutes). In most cases there is no reason to limit how often it is written and it is written anytime the library disconnects from the Q330. However, if continuity is written to a limited lifetime media such as flash this value can be used to reduce the number of writes that might be caused by a poor communications link.
 - word opt_dss_memory – If non-zero then limits the amount of memory (in kilobytes) that DSS (Data Subscription Service) can use. If set to zero, or if it is higher than the amount specified in the tokens, it is ignored. **7**

```
extern enum tliberr lib_unregister_ping (tcontext ct, tpar_register *rpar) ;
```

Similar to the register call but only does a "Ping" to the Q330 to see if it can be reached. Note that this is not an ICMP Ping, but uses Q330 format packets. The same "tpar_register" structure is used except you can leave the authorization code zero and the various timeouts like "opt_conntime" are not relevant. See a description of the state callback structure for the returned values. The state returns to "Idle" after completion.

```
extern void lib_change_state (tcontext ct, enum tlibstate newstate, enum tliberr reason) ;
```

This routine is used when you want to change from the current state. For instance to disconnect from the Q330 you could change the state to "LIBSTATE_IDLE". Once in that state and you have done whatever file handling is needed in the host then you could change the state to "LIBSTATE_TERM" to flush out any DP statistics to the host and terminate the station thread. Only after reaching "LIBSTATE_TERM" can you call:

```
extern enum tliberr lib_destroy_context (tcontext *ct) ; /* Return error if any */
```

To clean up the remaining data structures, normally before closing the host program.

The following can be used whether connected to the Q330 or not:

```
extern void lib_msg_add (tcontext ct, word msgcode, longword dt, string95 *msgsf) ;
```

The “msgcode” is from the list in libmsgs. Many messages also have additional information after the basic message, this is passed in “msgsf”. In fact, some messages have all their information in “msgsf”. For instance, libnetserver uses this call to provide information on it’s operation. If “dt” is non-zero then it will interpreted as seconds since 2000 and be used as an additional timestamp (in square brackets) in the message. Messages are queued and then sent to the LOG LCQ after a maximum of 100ms or before a internal library message is sent (to maintain correct order).

```
extern word lib_change_verbosity (tcontext ct, word newverb) ;
```

You can change the verbosity options while the thread is running using the above call. It will return the new value.

```
extern enum tlibstate lib_get_state (tcontext ct, enum tliberr *err, topstat *retopstat) ;
```

This routine will tell you the current state, the last error code, and a summary of operating status defined as:

```
typedef struct { /* operation status */
    string9 station_name ; /* network and station */
    word station_port ; /* data port number */
    longword station_tag ; /* tagid */
    t64 station_serial ; /* q330 serial number */
    longword station_reboot ; /* time of last reboot */
    longint timezone_offset ; /* seconds to adjust computer's clock */
    taccstats accstats ; /* accumulated statistics */
    word minutes_of_stats ; /* how many minutes of data available to make hour */
    word hours_of_stats ; /* how many hours of data available to make day */
    word auxinp ; /* bitmap of Aux. inputs */
    longint data_latency ; /* data latency in seconds or INVALID_LATENCY */
    longint status_latency ; /* seconds since received status from 330 or INVALID_LATENCY */
    longint runtime ; /* running time since current connection (+) or time it has been down (-) */
    longword totalgaps ; /* total number of data gaps since context created */
    single pkt_full ; /* percent of Q330 packet buffer full */
    word clock_qual ; /* Percent clock quality */
    longint clock_drift ; /* Clock drift from GPS in microseconds */
    integer mass_pos[6] ; /* mass positions */
    integer calibration_errors ; /* calibration error bitmap */
    integer sys_temp ; /* Q330 temperature in degrees C */
    single pwr_volt ; /* Q330 power supply voltage in volts */
    single pwr_cur ; /* Q330 power supply current in amps */
    longint gps_age ; /* age in seconds of last GPS clock update, -1 for never updated */
    enum tgps_stat gps_stat ; /* GPS Status */
    enum tgps_fix gps_fix ; /* GPS Fix */
    enum tpll_stat pll_stat ; /* PLL Status */
    double gps_lat ; /* Latitude */
    double gps_long ; /* Longitude */
    double gps_elev ; /* Elevation */
    tslidestat slidecopy ; /* sliding window status */
    longword last_data_time ; /* Latest data received, 0 for none */
    longword current_ip ; /* current IP Address of Q330 */
    word current_port ; /* current Q330 UDP Port */
} topstat ;
```

“station_name” is in format <network>-<station>, such as “TA-H08A”. “station_port” will be between zero (for data port 1) to three (for data port 4). “station_tag” should be what Kinometrics calls the “Serial Number” on the tag attached to the Q330. “station_reboot” will be in seconds since 2000. “timezone_offset” is the number of seconds required to be added to your host computer’s clock to get UTC.

“acstat” is actually a two dimensional array of values. The major index:

- AC_GAPS – Number of data gaps
- AC_BOOTS – Number of Q330 reboots
- AC_READ – Data received from Q330 in bytes per second (including IP and UDP headers)
- AC_WRITE – Data sent to the Q330 in bytes per second (including IP and UDP headers)
- AC_COMATP – Number of unsuccessful attempts to communicate with Q330
- AC_COMSUC – Number of successful attempts to communicate with the Q330
- AC_PACKETS – Number of Packets (complete packets, not bytes) received from the Q330
- AC_COMEFF – Communications Efficiency, a value of 1000 represents 100.0%
- AC_POCS – Number of POC's received (as provided by host)
- AC_NEWIP – Number of IP Address changes as a result of POC's
- AC_DUTY – Communications duty cycle (uptime), a value of 1000 represents 100.0%
- AC_THROUGH – Throughput, number of seconds of data received per second, a value of 100 = 1.00 seconds/second
- AC_MISSING – Data missing from the Q330, in seconds
- AC_FILL – Number of Fill (Flood) packets received
- AC_CMDTO – Number of command timeouts
- AC_SEQERR – Number of sequence errors on both the command and data ports.
- AC_CHECK – Number of checksum/CRC errors on both the command and data ports
- AC_IOERR – Number of I/O errors (serial ports)

The minor array index represents the time period:

- AD_MINUTE – During the last minute
- AD_HOUR – During the last hour, or “minutes_of_stats” minutes if it is less than 60
- AD_DAY – During the last day, or “hours_of_stats” hours if it is less than 24

Any of these entries may have the value of “INVALID_ENTRY” meaning it should be ignored.

“auxinp” is simply the bitmap of auxiliary inputs as reported by the Q330. Bit zero is apparently used in many installations to report sump pump status. “data_latency” and “status_latency” represent the time difference between the corrected host computer's clock and the last information received from the Q330. “INVALID_LATENCY” will be returned if the library has no way to calculate these values. “runtime” is positive if connected to the Q330 and indicates how many seconds the library has been connected. If “runtime” is negative it indicates how many seconds since the library has been disconnected from the Q330. “totalgaps” is an accumulation of the total number of gaps in the data from the Q330 regardless of time interval. “pkt_full” is the percentage of the packet buffer in the Q330 currently used for this data port.

“clock_qual” is the clock quality in percent based on the rules programmed into the tokens. “clock_drift” is the difference in microseconds between UTC time and the internal sampling. Normally within a few hundred microseconds except for very long GPS reception outages. “mass_pos” is an array with an entry for each of the mass position inputs on the Q330 (bit 0 = channel 1, etc). With no input connected the value is generally around 20 but has a range of -128 to +127. “calibration_errors” is a bitmap (bit 0 = channel 1) showing if any channels did not calibrate correctly at boot time.

“sys_temp” is the Q330's internal temperature in degrees Celsius. “pwr_volt” is the power supply voltage to the Q330 in volts. “pwr_cur” is the Q330 power supply current in amps. “gps_age” indicates the number of seconds since a reasonable time was processed from the GPS engine. “gps_stat” has the following values:

- GPS_OFF - Off
- GPS_OFF_LOCK - Off due to GPS Lock
- GPS_OFF_PLL - Off due to PLL Lock
- GPS_OFF_LIMIT - Off due to Time Limit
- GPS_OFF_CMD - Off due to Command
- GPS_ON - On
- GPS_ON_AUTO - On automatically
- GPS_ON_CMD - On by command
- GPS_COLDSTART - Cold-start

“gps_fix” has the following values:

- GPF_LF - Off, never locked
- GPF_OFF - Off, unknown lock
- GPF_1DF - Off, last fix 1D
- GPF_2DF - Off, last fix 2D
- GPF_3DF - Off, last fix 3D
- GPF_NL - On, never locked
- GPF_ON - On, unknown lock
- GPF_1D - 1D Fix
- GPF_2D - 2D Fix
- GPF_3D - 3D Fix
- GPF_NB - No GPS board

“pll_stat” has the following values:

- PLS_LOCK - Locked
- PLS_TRACK - Tracking
- PLS_HOLD - Hold
- PLS_OFF - Off

“gps_lat” is in degrees (positive is north). “gps_long” is in degrees (positive is east), and “gps_elev” is in meters.

“slidecopy” has a snapshot of the sliding window status, see the description for lib_get_slidestat for details.

“last_data_time” is the time of the latest data from the Q330 in seconds since 2000. It is zero if no data has been received since the thread was created. “current_ip” is the current IP Address being used to communicate with the Q330. This is normally the configured address but for stations with dynamic IP addresses it may change over time. Likewise, “current_port” is the UDP port number being used to communicate with the Q330.

```
extern enum tliberr lib_get_lcqstat (tcontext ct, tlcqstat *lcqstat) ;
```

This routine will give a snapshot of various parameters relating to what we like to call “Logical Channel Queues”. There is a queue entry for each piece of data generated (such as BHZ, ACE, LOG, etc.). If called when not in the “LIBSTATE_RUN” state then you will only receive the entries for the DP Statistics since they are the only ones that exist at that point. While in “LIBSTATE_RUN” you will also received those that are generated as a result of the data from the Q330. “tlcqstat” has the format:

```
typedef struct { /* format of one lcq status entry */
    string2 location ;
    byte chan_number ; /* channel number according to tokens */
    string3 channel ;
    longint rec_cnt ; /* number of records */
    longint rec_age ; /* number of seconds since update */
    longint rec_seq ; /* current record sequence */
    longint det_count ; /* number of detections */
    longint cal_count ; /* number of calibrations */
    longint arec_cnt ; /* number of archive new records */
    longint arec_over ; /* number of archive overwritten records */
    longint arec_age ; /* since last update */
    longint arec_seq ; /* current record sequence */
} tonelcqstat ;
typedef struct { /* format of the result */
    integer count ; /* number of valid entries */
    tonelcqstat entries[MAX_LCQ] ;
} tlcqstat ;
```

“count” has the number of valid entries. “location” can be anywhere from zero to two characters long, “channel” is always 3 characters. “chan_number” is a value from 0 to n for Q330 generated data and always 0xFF for DP Statistics. “rec_seq” and “arec_seq” indicates how many miniseed records have been generated (assuming continuity is valid) for the 512byte and archival miniseed records respectively. The other fields are “session” counts and ages, in other words since the “session” started. A session is since the most recent connection to the Q330 for Q330 generated data and since the station thread was started for the DP statistics. As a clarification “arec_over” is the number of times that an archival miniseed record has been updated with new data while “arec_cnt” is the number of new records.

Some other fun things to do while connected to the Q330:

```
extern void lib_ping_request (tcontext ct, tpingreq *ping_req) ;
```

This variation of the “Ping” is used when you are connected to the Q330. “tpingreq” has the format:

```
typedef struct { /* Ping request from host */
    word pingtype ; /* 0 for normal ping */
    word pingopt ;
    longword pingreqmap ;
} tpingreq ;
```

Only a type 0 “normal ping” is supported at this time. This is the format used by the unregistered ping call. You should set “pingopt” and “pingreqmap” to zero.

```
extern void lib_request_status (tcontext ct, longword bitmap, word interval) ;
```

The library always requests the following status from the Q330:

- SRB_LOGx – Data port status for the appropriate port
- SRB_GLB – Global status
- SRB_GST – GPS status
- SRB_BOOM – mass position and other state of health
- SRB_PLL – PLL status
- SRB_GSAT – GPS satellites, but only when needed for a new timing log entry.

The host can use this call to request further status by building a bitmap which will be “ored” with the status listed above. The status interval is also set using this routine.

```
extern enum tliberr lib_get_status (tcontext ct, longword bitnum, pointer buf) ;
```

This routine is used to get a copy of the specified status structure from the library. “bitnum” would be one of the SRB_XXX values. The host must provide the address of a buffer large enough to hold the status structure. The routine will return a value of “LIBERR_NOSTAT” if the status is not available. If not available then host should make sure it is in the list of automatic requests or specified in the lib_request_status bitmap. In addition to the automatically requested status above the following are supported by the library:

SRB_PWR – SMU Status **Q330**
 SRB_ARP - ARP Status
 SRB_SER1 - Serial Port 1 Status **Q330**
 SRB_SER2 - Serial Port 2 Status **Q330**
 SRB_SER3 - Serial Port 3 Status **Q330**
 SRB_ETH - Ethernet Status **Q330**
 SRB_BALER – Baler & Dialer Status **Q330** or Communications Status **Q335**
 SRB_DYN - Dynamic IP Address **Q330**
 SRB_AUX - Aux Board Status **Q330**
 SRB_SS - Serial Sensor Status **Q330**
 SRB_EP – Environmental Processor Status **Q330**
 SRB_FES – Front End Status **Q335**

```
extern enum tliberr lib_get_config (tcontext ct, longword bitnum, pointer buf) ;
```

Similar to the previous routine except the host is requesting a copy of a configuration structure. Returns “LIBERR_CFGWAIT” if the structure is not currently available. If not currently available the library will request it for the host and notify the host of it’s availability via the state callback.

CRB_GLOB - Global configuration
 CRB_FIX - Fixed configuration
 CRB_LOG - Logical Data port configuration
 CRB_GPSIDS - GPS ID's
 CRB_ROUTES – Routing Table **Q330**
 CRB_DEVS - CNP Devices **Q330**
 CRB_SENSCTRL - Sensor Control

```
extern enum tliberr lib_set_config (tcontext ct, longword bitnum, pointer buf) ;
```

Used to change configuration in the Q330. The only configuration that can be changed is the data port configuration using CRB_LOG. This is normally used to change communications link parameters but could also be used to modify the map of which main digitizer channels and frequencies are generated by the Q330 for that data port.

```
extern void lib_abort_command (tcontext ct) ;
```

Can be used to stop trying to send the current command to the Q330. This could be used if the host is notified by the state callback that the link is stalled (a command has been resent at least once without a response from the Q330). It removes the current command from the command queue.

```
extern enum tliberr lib_get_slidestat (tcontext ct, tslidestat *slidestat) ;
```

Can be used to generate a “sliding window” graphic display in the host program. Tslidestat is defined as:

```
typedef struct {
  word low_seq ; /* last packet number acked */
  word latest ; /* latest packet received */
  longword validmap[8] ;
} tslidestat ;
```

The library keeps a sliding window buffer of 256 packets, regardless of what the window size in use is, that is indexed using the least significant 8 bits from the 16 bit packet sequence number. “low_seq” is the last packet acked by the library (to the Q330) and “latest” is the latest packet received. Between those two is where the action is. “validmap” is a bitmap of which packets within those limits that the library has received (8 times 32 bits per longword is 256 bits).

```
extern void lib_send_usermessage (tcontext ct, string79 *umsg) ;
```

Allows the host to send a “User Message” to the Q330. These user message show up in message logs and one possible use is to make a note when system maintenance is done.

```
extern void lib_poc_received (tcontext ct, tpocmsg *poc) ;
```

A host uses this routine to update the library with information relevant to how to contact the Q330, usually as a result of a “SeisPOC” packet received from the Q330. “tpocmsg” has the format:

```
typedef struct { /* format of data provided by received POC */
  longword new_ip_address ; /* new dynamic IP address */
  word new_base_port ; /* port translation may have changed it */
  string95 log_info ; /* any additional information the POC receiver wants logged */
} tpocmsg ;
```

The library will update it’s internal copy of the Q330’s IP address and base port and then try to register with the Q330 if it is not currently registered. It will also add a message to the log with the host supplied “log_info”.

```
extern enum tliberr lib_get_commevents (tcontext ct, tcommevents *commevents) ;
```

A copy of the current “Comm Events” are returned to the host:

```
typedef struct { /* one comm event */
    char name[COMMLENGTH+1] ;
    boolean ison ;
} tonecomm ;
typedef tonecomm tcommevents[CE_MAX] ;
```

```
extern void lib_set_commevent (tcontext ct, integer number, boolean seton) ;
```

Allows the host to change the state of one Comm Event.

```
extern enum tliberr lib_get_detstat (tcontext ct, tdetstat *detstat) ;
```

Get the current state of the detectors. “tdetstat” is defined as:

```
typedef struct { /* format of one detector status entry */
    char name[DETECTOR_NAME_LENGTH + 11] ;
    boolean ison ; /* last record was detected on */
    boolean declared ; /* ison filtered with first */
    boolean first ; /* if this is the first detection after startup */
    boolean enabled ; /* currently enabled */
} tonedetstat ;
typedef struct { /* format of the result */
    integer count ; /* number of valid entries */
    tonedetstat entries[MAX_DETSTAT] ;
} tdetstat ;
```

“count” has the number of valid entries. Note in order to know what Q330 data stream the detector belongs to the detector name is in the format [<location>-]<seedname>:<detector-name>

```
extern void lib_change_enable (tcontext ct, tdetchange *detchange) ;
```

Allows the host to enable and disable detectors. “tdetchange” is defined as:

```
typedef struct { /* record to change detector enable */
    char name[DETECTOR_NAME_LENGTH + 11] ;
    boolean run_detector ;
} tdetchange ;
```

```
extern enum tliberr lib_get_ctrlstat (tcontext ct, tctrlstat *ctrlstat) ;
```

Returns a copy of the current control detector status. “tctrlstat” has the format:

```
typedef struct { /* format of one control detector status entry */
    string79 name ;
    boolean ison ; /* currently on */
} tonectrlstat ;
typedef struct { /* format of the result */
    integer count ; /* number of valid entries */
    tonectrlstat entries[MAX_CTRLSTAT] ;
} tctrlstat ;
```

“count” has the number of valid entries.

```
extern enum tliberr lib_get_dpcfg (tcontext ct, tdpcfg *dpcfg) ;
```

This is normally used when in “LIBSTATE_RUNWAIT” before the host requests the library change to the RUN state.

“tdpcfg” has the format:

```
typedef struct { /* format of essential items from tokens */
    string9 station_name ;
    word web_port ; /* web server TCP port */
    word net_port ; /* netserver (LISS) TCP port */
    word datas_port ; /* dataserver TCP port */
    longword webip ; /* IP address according to server challenge */
    tdss dss ; /* DSS configuration */
    tclock clock ; /* Clock configuration */
    word buffer_counts[MAX_LCQ] ; /* pre-event buffers + 1 */
} tdpcfg ;
```

“station_name” is in the format <network>-<station>. “web_port”, “net_port”, “datas_port”, “dss”, and “clock” are as read from the DP tokens. “webip” is actually the IP address of the host system as seen from the Q330’s point of view, which may differ from reality based on network translations. “buffer_counts” provides the number of 512byte buffers allocated per LCQ. This can be used as a “warning” to a host that should an earthquake occur it could receive that much data all at once from the LCQ. For instance, Pecos2 uses this information when it makes sure it has enough free data files available.

```
extern void lib_webadvertise (tcontext ct, string15 *stnname, string *dpaddr) ;
```

This routine is used if the host operates a web-server that should be advertised on the Q330’s web page. “stnname” must actually be no longer than 8 characters and is in the form of <network>-<station>. Older versions of Q330 software limited “dpaddr” to 24 characters and so you could only advertise in the form of <dotted decimal IP address>:<tcp port number>. Newer version allow up to 255 characters so you can advertise domain names. To tell if the Q330 can support the longer form you can check the fixed configuration structure: if fixed.flags and FF_NWEB < 0 {then supports longer form}

```
extern enum tliberr lib_send_tunneled (tcontext ct, byte cmd, byte response, pointer buf,
                                       integer req_size) ;
```

This routine is used if the host wishes to issue a command or status request that the library does not how to process. “cmd” is the command (for example C1_RQSP would be 0x2D) to send and “response” is what the expected response is (for example C1_SPP would be 0xAE). “buf” is the address of a buffer containing the “payload” for the command and “req_size” is the size of that payload. If the library has already sent a tunneled command and the response has not been received then this routine will return “LIBERR_TUNBUSY”.


```
extern enum tliberr lib_get_tunneled (tcontext ct, byte *response, pointer buf,
                                       integer *resp_size) ;
```

After sending a tunneled command this routine is used to get the result. “response” will be the actual response received from the Q330. “buf” is a pointer to a buffer that the host must provide that is large enough for the response. “resp_size” will be set to the actual size of the payload received. If the response has not yet been received then this routine will return “LIBERR_TUNBUSY”.


```
extern enum tliberr lib_conntiming (tcontext ct, tconntiming *conntiming, boolean setter) ;
```

This call allows changing most of the timeouts in the library to suite specific needs. For instance in a baler application it might setup the defaults for continuous operation and if it finds out it is power-cycled then change them. If “setter” is zero then it will return the current values in “conntiming” else the values in “conntiming” will be written to the library. tconntiming has the following format:


```
typedef struct { /* format of lib_change_conntiming */
    word opt_conntime ; /* maximum connection time in minutes if non-zero */
    word opt_connwait ; /* wait this many minutes after connection time or buflevel shutdown */
    word opt_buflevel ; /* terminate connection when buffer level reaches this value if non-zero */
    word data_timeout ; /* timeout in minutes for data timeout (default is 10) */
    word data_timeout_retry ; /* minutes to wait after data timeout (default is 10) */
    word status_timeout ; /* timeout in minutes for status timeout (default is 5) */
    word status_timeout_retry ; /* minutes to wait after status timeout (default is 5) */
    word piu_retry ; /* minutes to wait after port in use timeout (default is 5) */
} tconntiming ;
```

```
extern longint lib_crccalc (tcontext ct, pbyte p, longint len) ; 
```

Allows usage of the library CRC calculation routine for other uses. “p” points to a series of bytes as input. “len” is the number of bytes in the input. Returns the CRC.

```
extern enum tliberr lib_send_checkip (tcontext ct, longword ip) ; 
```

Generates a C2_REGRESP packet to the Q330. The response is returned via the baler callback.

```
extern enum tliberr lib_md5_operation (tcontext ct, tmd5op *md5op) ; 
```

Allows usage of the library MD5 calculation routine for other uses. “tmd5op” has the following format:

```
enum tmd5op_type {MDO_INIT, /* initialize buffer */
                  MDO_UPDATE, /* Update acc */
                  MDO_RESULT} ; /* Return result */
typedef struct { /* MD5 Operations */
    enum tmd5op_type otype ; /* operation to do */
    pbyte ptr ; /* pointer to input data */
    integer cnt ; /* length of input data */
    t128 res ; /* result */
} tmd5op ;
```

```
extern enum tliberr lib_set_access_timer (tcontext ct, word seconds) ;
```

Only used in “Baler Mode”. Used to keep the library from disconnecting from the Q330 for the specified number of “seconds”.

```
extern enum tliberr lib_set_freeze_timer (tcontext ct, integer seconds) ;
```

If seconds is non-zero then it the library ignores data packets from the Q330 for that many seconds.

```
extern enum tliberr lib_flush_data (tcontext ct) ;
```

Causes the library to generate SEED records with whatever data is currently available, both Q330 and DP LCQs. In normal operation Q330 LCQs are flushed when disconnecting from the Q330 and DP LCQs are flushed right before the station thread terminates.

```
#ifndef OMIT_SERIAL
extern enum tliberr lib_inject_packet (tcontext ct, pbyte payload, byte protocol,
                                     longword srcaddr, longword destaddr, word srcport, word destport,
                                     word datalength, longword seq, longword ack, word window, byte flags) ;
#endif
```

Allows a client program or other library to use the serial interface handling routines and SLIP encoder in the library when connected to a Q330 over the serial port. In the case of “Baler Mode” the baler library can use this routine to send web server responses (to requests received from the baler callback) to route through the Q330 to the internet. Only UDP and TCP protocols are supported. “payload” must point to a structure that has room for the IP and UDP/TCP headers before the payload.

Callbacks

void state_callback (pointer p)

This callback is used for multiple functions. “p” points to the following structure:

```
typedef struct { /* format for state callback */
    tcontext context ;
    enum tstate_type state_type ; /* reason for this message */
    string9 station_name ;
    longword subtype ; /* to further narrow it down */
    longword info ; /* new highest message for ST_MSG, new state for ST_STATE,
                    or new status available for ST_STATUS */
} tstate_call ;
```

“context” is the station thread context. “state_type” is one of the following:

- ST_STATE – Change in state, new state is in “info”. 8 Subtype is 1 when connected to a Q335 or 0 if connected to a Q330. This is only valid only after the state has progressed past LIBSTATE_READCFG.
- ST_STATUS – New Q330 status available, bitmap of currently available status in “info”.
- ST_CFG – New Q330 configuration available, bitmap of currently available configuration in “info”.
- ST_STALL – Change in stalled communications link status. “info” is one for stalled link, zero for non-stalled.
- ST_PING – Ping result. “subtype” has the ping response type (1). “info” has the round-trip time in milliseconds or 0xFFFFFFFF if there was no response in 5 seconds.
- ST_TICK – A new second of Q330 data has been received. “info” has the seconds since 2000. “subtype” has the microseconds offset (interpret as a signed 32 bit value).
- ST_OPSTAT – Operational status has been updated (once per minute).
- ST_TUNNEL – Your tunneled command response is now available, have a nice day.

void msg_callback (pointer p)

This callback is used when the library generates a new message. “p” points to the following structure:

```
typedef struct { /* format for messages callback */
    tcontext context ;
    longword msgcount ; /* number of messages */
    word code ;
    longword timestamp, datetime ;
    string95 suffix ;
} tmsg_call ;
```

“context” is the station thread context. “msgcount” is the new total messages (the first message will have msgcount = 1). “code” is the message code (such as LIBMSG_SEQRESUME) and the host would normally use lib_get_msg in the libmsgs file to translate to text. “timestamp” is the seconds since 2000 indicating when the message was generated according to the host computer’s adjusted clock. “datetime”, if non-zero, indicates the seconds since 2000 of the data this message is in reference to. “suffix” is additional information beyond what is contained by the code.

For messages with a zero “datetime” this would be an example of how the library would generate a message for the miniseed message log:

2006-09-04 00:23:53 {101} Msg From 216.120.82.59:Baler14-1.95 tag 12345: XX-BART reg 2006-09-04 00:23:52

And for a message with a non-zero “datetime”:

2006-09-04 00:19:22 {302}[2006-09-04 00:00:00] Saving Configuration to Backup Memory

The first time is when the message was logged, the number between the curly braces is the message code, and the time between the square brackets is the data time for when that message was generated.

void one_second_callback (pointer p)

This callback is used to pass one second data to the host. The number of samples included and how often this callback is used is of course dependent on the actual sampling rate per channel. “p” points to the following structure:

```
typedef struct { /* for 1 second and low latency callback */
    longword total_size ; /* number of bytes in buffer passed */
    tcontext context ;
    string9 station_name ; /* network and station */
    string2 location ;
    byte chan_number ; /* channel number according to tokens */
    string3 channel ;
    word padding ;
    integer rate ; /* sampling rate */
    longword cl_session ; /* closed loop session number */
    longword reserved ; /* must be zero */
    double cl_offset ; /* closed loop time offset */
    double timestamp ; /* Time of data, corrected for any filtering */
    word filter_bits ; /* OSF_xxx bits */
    word qual_perc ; /* time quality percentage */
    word activity_flags ; /* same as in Miniseed */
    word io_flags ; /* same as in Miniseed */
    word data_quality_flags ; /* same as in Miniseed */
    byte src_channel ; /* source blockette channel */
    byte src_subchan ; /* source blockette sub-channel */
    longint samples[MAX_RATE] ; /* decompressed samples */
} tonesec_call ;
```

“total_size” indicates the total size of the structure that is used. Since the structure has to handle up to 200 samples per second in most cases the structure will be smaller. In practice this probably isn’t relevant but for Seneca it writes these packets to disk and I wrote a program to convert the packets into uncompressed Steim1 miniseed to allow verification. “context” is the station thread context.

“station_name” is in the format <network>-<station>. “location” can be anywhere from zero to two characters long, “channel” is always 3 characters. “chan_number” is a value from 0 to n for Q330 generated data and always 0xFF for DP Statistics. “rate” is the sampling rate in samples per second if positive or seconds per sample if negative. “cl_session” and “cl_offset” are not currently used. “timestamp” is starting time of the data in seconds since 2000 and has already been corrected for any filter delays.

“filter_bits” is any combination of the OSF_xxx values indicating why the data was sent to the host. “qual_perc” is the current clock quality. “activity_flags” can have the Seed Event-in-progress and Calibration-in-progress bits on. “io_flags” can have the clock-locked flag on. “data_quality_flags” can have the questionable-timetag flag on.

“src_channel” and “src_subchan” are the channel and sub-channel values setup in the DP tokens and relate to the blockettes generated by the Q330 and sent to the DP. In most cases a DP wouldn’t care about those values and instead would go solely by the Seed channel name.

“samples” is an array of decompressed samples with the number of samples equal to the “rate” if rate is positive or one sample for sub-Hz sampling rates. The host needs to make a copy of the valid parts of the structure before returning. The host must not block in the callback for any significant amount of time.


```
void miniseed_callback (pointer p)
```

and

```
void archival_miniseed_callback (pointer p)
```

are the callback routines for 512byte miniseed and archival miniseed data. In both cases “p” points to the following structure:

```
typedef struct { /* format for miniseed and archival miniseed */
    tcontext context ;
    string9 station_name ; /* network and station */
    string2 location ;
    byte chan_number ; /* channel number according to tokens */
    string3 channel ;
    integer rate ; /* sampling rate */
    longword cl_session ; /* closed loop session number */
    double cl_offset ; /* closed loop time offset */
    double timestamp ; /* Time of data, corrected for any filtering */
    word filter_bits ; /* OMF_xxx bits */
    enum tpacket_class packet_class ; /* type of record */
    enum tminiseed_action miniseed_action ; /* what this packet represents */
    word data_size ; /* size of actual miniseed data */
    pointer data_address ; /* pointer to miniseed record */
} tminiseed_call ;
```

“context” is the station thread context. “station_name” is in the format <network>-<station>. “location” can be anywhere from zero to two characters long, “channel” is always 3 characters. “chan_number” is a value from 0 to n for Q330 generated data and always 0xFF for DP Statistics. “rate” is the sampling rate in samples per second if positive or seconds per sample if negative. “cl_session” and “cl_offset” are not currently used. “timestamp” is starting time of the data in seconds since 2000 and has already been corrected for any filter delays.

“filter_bits” is any combination of the OMF_xxx values indicating why the data was sent to the host. “packet_class” indicates the type of Seed data:

```
enum tpacket_class {PKC_DATA, PKC_EVENT, PKC_CALIBRATE, PKC_TIMING, PKC_MESSAGE, PKC_OPAQUE} ;
```

“PKC_DATA” is used for all time-series data and consists of the Seed data header, a blockette 1000, a blockette 1001, and multiple 64byte Steim 2 compressed frames. If the “mini_embed” field in tpar_create was set non-zero then event and calibration blockettes may be present preceeding the compressed data frames. “PKC_EVENT” and “PCK_CALIBRATE” are only issued if the “mini_separate” field in tpar_create was set non-zero. These simply have the Seed data header, a blockette 1000, and one event or calibration blockette. “PKC_TIMING” contain a Seed data header, a blockette 1000, and one or more (possible in archival miniseed) timing blockettes. “PKC_MESSAGE” contain a Seed data header, a blockette 1000, and one or more lines of text (terminated by a carriage return and line feed). “PKC_OPAQUE” contain a Seed data header, a blockette 1000, and one or more opaque blockettes.

“miniseed_action” tells the host what should be done:

```
enum tminiseed_action {MSA_512, /* new 512 byte packet */
    MSA_ARC, /* new archival packet, non-incremental */
    MSA_FIRST, /* new archival packet, incremental */
    MSA_INC, /* incremental update to archival packet */
    MSA_FINAL, /* final incremental update */
    MSA_GETARC, /* request for last archival packet written */
    MSA_RETARC} ; /* client is returning last packet written */
```

“MSA_512” is the only action for the miniseed callback, all others are for the archival miniseed callback. “MSA_512” means the packet contains a new 512byte miniseed record for the host.

“MSA_ARC” and “MSA_FIRST” both indicate a new archival miniseed record has been generated. The difference is that MSA_FIRST means that it could be added onto in the future using “MSA_INC” with the last update being action “MSA_FINAL”. This mechanism is used for data streams with a sample rate up to the value specified by the “amini_512highest” field in tpar_create so that archival miniseed can have less latency, especially for low rate data. You can defeat this capability by setting “amini_512highest” to a very low value, such as -1000.

“MSA_GETARC” is a request from the library to provide it with the last archival miniseed record that the host wrote. If this is available then the host copies that record into the data buffer provided by the library (via “data_address”) and then the host changes “miniseed_action” to “MSA_RETARC”. For instance this feature is used in Pecos2 so when a new connection is established with the Q330 the library can add onto the last 4K Seed record (if time contiguous) rather than wasting space by starting a new one.

“data_size” indicates the size of the data, it will always be 512 for the miniseed callback and whatever was specified for the archival miniseed callback. “data_address” points to a buffer holding the data. The host needs to make a copy of the structure before returning. The host must not block in the callback for any significant amount of time.

Data Extension Blockette Flags **11**

For a Q330 the deb_flags field is:

- Bits 0 - 2 are set to the value 6. This indicates the data source is a Q330.
- Bit 6 is set.
- Bit 7 is set for a event-only data stream.

For a Q335 the deb_flags field is:

- Bits 0 and 1 indicate the PGA gain as a power of 2 (00 = 1, 01 = 2, 10 = 4, 11 = 8)
- Bit 2 if set indicates an additional gain multiplier of 8 due to using the low voltage input option.
- Bit 5 is set to indicate the new encoding.
- Bit 7 is set for event-only data stream.

void baler_callback (pointer p)

Is the callback to handle “Baler Mode” related functions. “p” points to the structure:

```
enum tbaler_type {BT_Q330TIME, /* number of seconds since 2000 from Q330 */
                 BT_UDPRECV, /* UDP packet received that might be for baler */
                 BT_TCPRECV, /* TCP packet received that might be for baler */
                 BT_REGRESP, /* Registration response */
                 BT_TIMER, /* 100ms timer */
                 BT_BACK, /* Baler Acknowledge */
                 BT_SOCKET, /* Opening a baler socket */
                 BT_BACK335} ; /* Baler Acknowledge for Q335 - not currently implemented */ 11
enum tbaler_socket {BS_CONTROL, BS_DATA, BS_BCASTCTRL} ;
typedef struct { /* format for baler callback */
    tcontext context ;
    enum tbaler_type baler_type ; /* reason for this message */
    string9 station_name ;
    longword response ; /* Some calls require a response */
    longword info ; /* A 32 bit value depending on callback */
    void *info2 ; /* Decoded IP header or address of tback packet */
    void *info3 ; /* Decoded UDP or TCP Header */
    void *info4 ; /* Address of UDP or TCP payload, info has payload length */
} tbaler_call ;
```

For “BT_Q330TIME” info has the number of seconds since 2000 from the global status packet from the Q330. “BT_UDPRECV” is sent if a UDP packet is received by the library over the serial port from the Q330 that either is not for the library’s IP address or not for the control or data UDP port. “info” has the length of the UDP payload, “info2” has the address of the IP header, “info3” has the address of the UDP header, and “info4” has the address of the payload. Likewise, “BT_TCPRECV” is sent if a TCP packet is received by the library over the serial port from the Q330.

“BT_REGRESP” is sent if the C2_REGCHK packet is received from the Q330 with the magic number in “info”. “BT_TIMER” is sent every 100ms and can be used for timeouts in a client or other library. “BT_BACK” is sent when a C2_BACK packet is received from the Q330. “info2” has a pointer to the C2_BACK packet (past the QDP header) and “info” has the contents of the point-of-contact/baler-ip-address field. “BT_SOCKET” is sent when an Ethernet socket is opened to talk to a Q330. This can be used by the client or other library to do any special handling required. “BS_CONTROL” is used when opening the control socket in normal mode. “BS_DATA” is for the data socket. “BS_CASTCTRL” is used when opening the control socket for doing the baler announce operation which must use broadcast mode. The BS_xxxx value is put into “info” and the socket path is put into “info2”.

```
void file_callback (pointer p)
```

For backwards compatibility if the “file_owner” field in the thread creation structure is not set then file operations in libsupport are done as in previous versions. If “file_owner” is set it must point to the following structure:

```
typedef struct { /* for file access */
    tcallback call_fileacc ; /* File access callback */
    pointer station_ptr ; /* opaque pointer */
} tfile_owner ;
typedef tfile_owner *pfile_owner ;
```

“call_fileacc” must be set and point to the file_callback routine. “station_ptr” is optional and can be used by a client to determine which station thread is issuing the callback but is not used in any way by the library.

Pointer “p” points to the following structure:

```
enum tfileacc_type {FAT_OPEN,          /* Open File */
                   FAT_CLOSE,         /* Close File */
                   FAT_DEL,           /* Delete File */
                   FAT_SEEK,          /* Seek in File */
                   FAT_READ,          /* Read from File */
                   FAT_WRITE,         /* Write to File */
                   FAT_SIZE,          /* Return File size */
                   FAT_CLRDIR,        /* Clear Directory */
                   FAT_DIRFIRST,      /* Get first entry in directory */
                   FAT_DIRNEXT,       /* Following entries, -1 file handle = done */
                   FAT_DIRCLOSE}; /* If user wants to stop the scan before done */
typedef struct { /* format of file access callback */
    pfile_owner owner ; /* information to locate station */
    enum tfileacc_type fileacc_type ; /* reason for this message */
    integer response ; /* -1 or file handle for open */
    pointer fname ; /* pointer to filename for open & delete, address of buffer read/write */
    integer opt1 ; /* first open parameter or descriptor */
    integer opt2 ; /* second open parameter if needed, or read/write size */
} tfileacc_call ;
```

“owner” is the “file_owner” from the thread creation parameter if the call is generated by the library (continuity calls) or whatever pointer was provided by a client or other library when using the routines in libsupport. The structure fields have different uses depending “fileacc_type”:

- FAT_OPEN – “fname” is a pointer to the file name. For WIN32 “opt1” is the read-write mode and “opt2” is the open-create mode. For Posix “opt1” are the permissions and “opt2” are the read-write and open-create modes. “response” must set to the file descriptor if successful or –1 for a failure.
- FAT_CLOSE – “opt1” is the file descriptor.
- FAT_DEL – “fname” is a pointer to the file name.
- FAT_SEEK – “opt1” is the file descriptor and “opt2” is the file offset in bytes. “response” is zero if no error or non-zero for error.
- FAT_READ – “opt1” is the file descriptor, “opt2” is the number of bytes, and “fname” is a pointer to the destination buffer. “response” is zero if no error or non-zero for error.
- FAT_WRITE – “opt1” is the file descriptor, “opt2” is the number of bytes, and “fname” is a pointer to the source buffer. “response” is zero if no error or non-zero for error.

- FAT_SIZE – “opt1” is the file descriptor. “response” is the file size in bytes.
- FAT_CLRDIR – “fname” is a pointer to the directory path name. All files within that directory should be removed.
- FAT_DIRFIRST – “fname” is a pointer to the directory path name. The first file is opened for reading and “response” is set to the file descriptor if successful or -1 for a failure. The client must call FAT_CLOSE once it is done with the file and before it tries to open another one.
- FAT_DIRNEXT – Same as FAT_DIRFIRST except it opens the next file in the directory.
- FAT_DIRCLOSE – Terminates directory searching early (without waiting for failure). **5**

Adding additional FIR Filters

As noted in the section about lib_create_context a host can add additional FIR filters beyond the standard three in the library. A linked list of filters must be created using the following structure:

```
typedef struct tfilter { /* coefficient storage for FIR filters */
    struct tfilter *link ; /* list link */
    char fname[FILTER_NAME_LENGTH + 1] ; /* name of this FIR filter */
    byte fir_num ; /* filter number */
    double coef[FIRMAXSIZE] ; /* IEEE f.p. coef's */
    longint len ; /* actual length of filter */
    double gain ; /* gain factor of filter */
    double dly ; /* delay in samples */
    longint dec ; /* decimation factor of filter */
} tfilter ;
```

Make sure “link” is NIL/NULL for the last filter in your custom list. “fname” must match whatever the name in the DP tokens. “fir_num” must be zero. “coef” is an array that contains the actual filter coefficients. “len” is the number of coefficients in the filter and of course must not be higher than “FIRMAXSIZE”. “gain” is for informational purposes, the host must have already scaled the coefficients by this factor when it puts them into “coef”, the library does not do this. “dly” indicates the filter delay in samples, the library will use this to calculate a filter delay in seconds based on the sample rate. “dec” is the decimation factor which the library will use to calculate the output sample rate of the filter based on it and the input sample rate.

Routines in Libmsgs

These routines require the address of a buffer “result” to store the result. “result” is also returned as the function result which can be useful in some circumstances, such as a string parameter to sprintf.

```
extern char *lib_get_msg (word code, string95 *result) ;
```

Converts a message code (LIBMSG_xxx) into a string

```
extern char *lib_get_errstr (enum tliberr err, string63 *result) ;
```

Converts an error code (LIBERR_xxx) into a string

```
extern char *lib_get_statestr (enum tlibstate state, string63 *result) ;
```

Converts a library state (LIBSTATE_xxx) into a string

```
extern char *showdot (longword num, string15 *result) ;
```

Converts a IPV4 address to a dotted-decimal string

```
extern char *command_name (byte cmd, string95 *result) ;
```

Converts a QDP command value into a string

```
extern char *lib_gps_state (enum tgps_stat gs, string63 *result) ;
```

Converts a GPS State (GPS_xxx) into a string

```
extern char *lib_gps_fix (enum tgps_fix gf, string63 *result) ;
```

Converts a GPS Fix (GPF_xxx) into a string

```
extern char *lib_pll_state (enum tpll_state ps, string31 *result) ;
```

Converts a PLL State (PLL_xxx) into a string

```
extern char *lib_acc_types (enum tacctype, string31 *result) ;
```

Converts an accumulated status type (as in topstat) into a string

Routines in Libseed

The following may be of some use to a host program:

```
extern char *seed2string(tlocation *loc, tseed_name *sn, string15 *result) ;
```

Converts a location/seed channel name pair into a string. If location is not spaces then the result will be in the form of <location>-<seedname> else it generates <seedname>

```
extern double extract_time (tseed_time *st, byte usec) ;
```

Given a Seed time structure and a microseconds offset (from blockette 1001) generate a double precision seconds since 2000.

The following routines take a pointer into a Seed record and extract a data structure into host native format.

```
extern void loadblkhdr (pbyte *p, blk_min *blk) ;
```

Load a blockette header (type and link)

```
extern void loadtime (pbyte *p, tseed_time *seedtime) ;
```

Load the Seed time structure

```
extern void loadseedhdr (pbyte *psrc, seed_header *hdr, boolean hasdeb) ;
```

Load the Seed data header, set "hasdeb" non-zero if the source has a blockette 1001

```
extern void loadtiming (pbyte psrc, timing *tim) ;
```

Load a timing blockette

```
extern void loadmurdock (pbyte psrc, murdock_detect *mdet) ;
```

Load a Murdock-Hutt detection blockette

```
extern void loadthreshold (pbyte psrc, threshold_detect *tdet) ;
```

Load a threshold detection blockette

```
extern void loadstep (pbyte psrc, step_calibration *stepcal) ;
```

Load a step calibration blockette

```
extern void loadsine (pbyte psrc, sine_calibration *sinecal) ;
```

Load a sine calibration blockette

```
extern void loadrandom (pbyte psrc, random_calibration *randcal) ;
```

Load a random calibration blockette

```
extern void loadabort (pbyte psrc, abort_calibration *abortcal) ;
```

Load a calibration abort blockette

```
extern void loadopaquehdr (pbyte psrc, opaque_hdr *ophdr) ;
```

Load an opaque blockette header

Routines in Libnetserv

```
extern pointer lib_ns_start (tns_par *nspar) ;
```

Start a netserver thread and open a socket. If no error it returns a pointer to it's "context", else a NIL pointer. "tns_par" has the following format:

```
typedef struct {
    longword lowip, highip ;
} twhitelist ;
typedef struct { /* creation parameters for one netserver */
    word ns_port ; /* TCP port number */
    integer server_number ;
    integer whitecount ;
    integer sync_time ;
    integer record_count ; /* number of records allocated */
    pointer stnctx ; /* station context */
    tnsbuf *nsbuf ; /* pointer to circular buffer */
    twhitelist whitelist[MAX_NETWHITE] ;
} tns_par ;
```

"ns_port" is the TCP the netserver is to listen on. "server_number" is a value from 1 to the number of netservers that will be run for this station. "whitecount" indicates how many entries in "whitelist" are valid. "sync_time", if non-zero indicates that if there have been no packets sent since that many seconds then a sync packet is to be sent to the client to keep the link alive.

"record_count" is the number of 512byte records that will fit into the buffer that the host has allocated. "stnctx" is the station thread context for the parent station. "nsbuf" is the address of the buffer allocated by the host. "whitelist" is a list of IP address ranges that will be accepted by the netserver. If "whitecount" is zero then connections will be accepted from any IP address.

```
extern void lib_ns_stop (pointer ct) ;
```

Stops the netserver and closes it's socket(s).

```
extern void lib_ns_send (pointer ct, pcompleted_record *pbuf) ;
```

Adds a new 512byte record to the circular queue. If the queue is full then the oldest record is removed to make room for the new record.

Routines in Libpoc *(not to be confused with Lompoc)*

extern pointer lib_poc_start (tpoc_par *pp) ;

Starts a SeisPOC receiver thread and opens a socket. If no error it returns a pointer to it's "context". "tpoc_par" has the format:

```
typedef struct { /* parameters for POC receiver */
    word poc_port ; /* UDP port to listen on */
    tpocproc poc_callback ; /* procedure to call when poc received */
} tpoc_par ;
```

"poc_port" is the UDP port to listen on. Port 2254 has been assigned by IANA (Internet Assigned Numbers Authority) for these messages. "poc_callback" is the address of a callback routine of the format:

void callback_routine (tpocstate pocstate, tpoc_recvd poc_recv)

"pocstate" is "PS_NEWPOC" if a new POC has been received in which case "poc_recv" is valid. It is "PS_CONNRESET" if the socket received a connection reset error from the socket for some reason. "tpoc_recvd" has the following format:

```
typedef struct { /* Format of callback parameter */
    t64 serial_number ; /* Q330's serial port */
    longword ip_address ; /* new dynamic IP address */
    word base_port ; /* port translation may have changed it */
    word data_port ; /* for data port */
} tpoc_recvd ;
```

"serial_number" is the serial number of the Q330 and "data_port" is the data port for which the POC was generated. The host should use these two pieces of information to locate the appropriate station thread for which this POC should be dispatched. Once this is done the host can call the "lib_poc_received" routine for that station with the "ip_address" and "base_port" information.

extern void lib_poc_stop (pointer ctr) ;

Terminate the POC receiver thread and close the socket.

Routines in Libsupport

extern pointer extend_link (pointer base, pointer add) ;

Generic linked list expansion. Returns new base pointer.

extern void zpad (pchar s, integer lth) ;

Add zeroes to the beginning of the string to take up "lth" characters.

extern double now (void) ;

Returns current host system time in seconds (and fraction) since 2000-01-01 00:00:00. This time is important because this is the base time reference used in the Q330 software.

extern word day_julian (word yr, word wmonth, word day) ;

Convert Gregorian year, month, and day to julian day.

extern longint lib330_julian (tsystemtime *greg) ;

Convert Gregorian date and time into seconds since 2000.

```
extern void day_gregorian (word yr, word jday, word *mth, word *day) ;
```

Convert year and julian day to Gregorian month and day.

```
extern void jul_string (longint jul, string *result) ;
```

Convert seconds since 2000 into string in format yyyy-mm-dd hh:mm:ss

```
extern void lib330_gregorian (longint jul, tsystemtime *greg) ;
```

Convert seconds since 2000 into Gregorian date and time

```
extern longword getip (string *s, boolean *domain) ;
```

Convert dotted decimal IPV4 address or domain name into IP address. “domain” set TRUE if required a domain name lookup.

```
extern char *lib330_upper (pchar s) ; 7
```

Converts a string to upper case.

The following file access routines use “tfile_handle” as a file descriptor. The actual type is defined in the platform header file. Also defined in platform is the constant “INVALID_FILE_HANDLE” which is also dependent on the operating system. All routines have a new optional pointer as the first parameter. For a discussion of the pointer see the file callback description earlier. If you want the library to do all the file functions without doing the callback then just use NIL (NULL) for the pointer.

```
extern tfile_handle lib_file_open (pfile_owner powner, string *path, integer mode) ;
```

Opens/Creates a file and returns a file descriptor handle or INVALID_FILE_HANDLE if error. These file routines are very simple and are intended for use internally for handling of the continuity files. “mode” is the “oring” of the following:

- LFO_CREATE – Create new file, overwrite existing file
- LFO_OPEN – Open existing file
- LFO_READ – Allow reading file
- LFO_WRITE – Allow writing file

```
extern void lib_file_close (pfile_owner powner, tfile_handle desc) ;
```

Close file.

```
extern boolean lib_file_seek (pfile_owner powner, tfile_handle desc, integer offset) ;
```

Seek to byte “offset” in file, returns TRUE if error.

```
extern boolean lib_file_read (pfile_owner powner, tfile_handle desc, pointer buf, integer size) ;
```

Read “size” bytes from file into “buf”, returns TRUE if error.

```
extern boolean lib_file_write (pfile_owner powner, tfile_handle desc, pointer buf, integer size) ;
```

Write “size” bytes to file from “buf”, returns TRUE if error.

```
extern void lib_file_delete (pfile_owner powner, string *path) ;
```

Delete file by name.

```
extern integer lib_file_size (pfile_owner powner, tfile_handle desc) ;
```

Returns the size in bytes of the file.