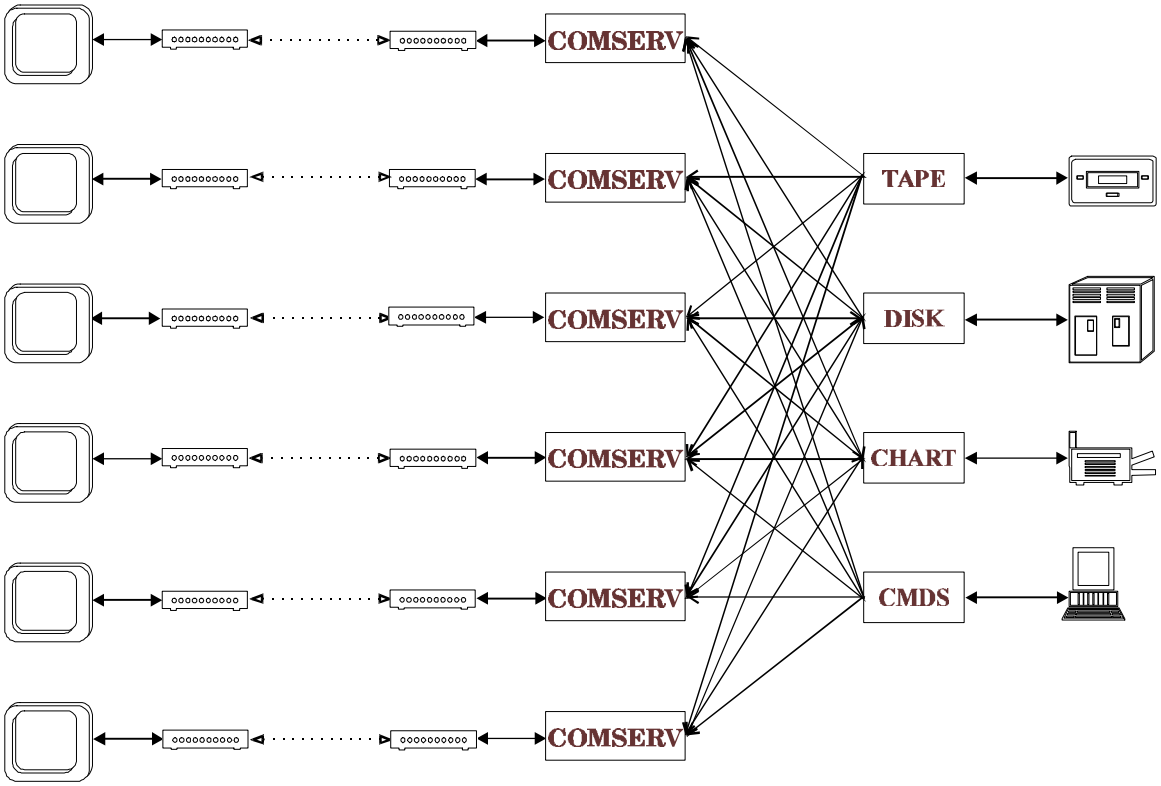
COMSERV USER'S GUIDE
16 JUNE 1997

This document describes the interface, on Sun Unix and OS-9/68K systems, to the Communications Server (Comserv) process. The Comserv process interfaces through a serial or ethernet link to one Quanterra Station. There may be multiple Comserv processes running, identified by the Station Name (4 characters maximum, padded on the right with spaces). Each client is identified by it's name (4 characters maximum, padded on the right with spaces). This document corresponds to the MShear-B4 release.

System Organization

Comserv and associated libraries available to clients provide for a software "crossbar switch" to allow all clients access to all stations (or any combinations of stations as the client requires).



Conventions :

- All parameters, function return values, and record fields are 32 bit signed integers unless otherwise noted.
- All procedure names, types, and variables are lower case only, but may contain underscores for readability.
- All constants are upper case only.
- Hexadecimal numbers are preceded with an dollar ($) sign, such as $7F. For you C programmers, this would be 0x7F.
- A specification such as "String15" means a byte containing the number of valid characters, followed by up to 15 characters. This is "industry standard Pascal" format. A specification such as "char15" means a null terminated string of 15 characters maximum which is the C standard. Both require 16 bytes of memory. Comserv does as little in the way of conversions as possible and since the DA is programmed in Pascal, that string format is retained. Utility routines in "stuff.c" provide conversions between formats when required.
- It should be noted that station and client names are converted internally to longintegers for speed of comparison (since C does not support array comparisons without using a library routine).
- The "int" data type should only be used for operating system specific variables and parameters since the size of "int" is 16 bits on some systems and 32 bits on other systems. Non-OS specific variables should use short or long integers to help in transportability.

Configuration files :

The master station file is called /r0/stations.ini on OS-9 systems, and /etc/stations.ini on UNIX systems. The master station file contains entries in the following format :

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

[NAME]
etc. etc.
```

Where "NAME" is the station name and must be 4 characters or less. The station directory contains a file called "station.ini" that contains station specific information for Comserv and other programs. Comserv will not recognize any station that does not have "source=comlink".

If this is a station that uses comserv then the station.ini file must have the comlink section, with optional lines in italics :

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

Port is the path name for the serial port and baud is the speed (normally 9600, 19200, or 38400). Parity is optional, and if not specified is no parity, which is Quanterra standard. If port is not specified, then ipport must be specified and is the TCP or UDP port number to use, this number must be between 5000 and 65535. udpaddr is the IP address of the DA when using UDP packets, which is recommended. If for some reason you are getting excessive sequence errors TCP operation can be selected by not specifying the udpaddr parameter.

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

Override is no by default, enabling station name checking from the DA, an error is generated if the station name in the packets does not agree with the station name you think you are talking to. If override is set to yes, then the station name in the packets will be replaced (useful for internal testing). Station can be used to override the station name given on the Comserv command line, the point in doing this is left as an exercise for the reader.

Notify is no by default, setting to yes will enable the LINK_PKT/ULTRA_REQ handshaking required when notify is enabled on the DA. flow is no by default, setting to yes might enable RTS/CTS hardware handshaking on a serial port, should you system actually support it.

Shear and Ultra Shear DA systems do not encode the seed name and location for message and timing log, while Multi-Shear does.. If the DA is running Multi-Shear you can set the seedin flag to yes, Comserv will then use the embedded names. If not, Log_seed can be used to override the default Seed name of "LOG" (with no location) for messages and Timing_seed can be used to override the default Seed name of "ACE" (with no location) for timing blockettes.

Segid is used for the shared memory segment for that comserv. It must be unique on the system. Pollusecs defaults to 50000 and is the number of microseconds that the comserv process "sleeps" between looking for new data from the serial port or for service requests from clients. This is a tradeoff between system overhead and serial port/service time.

Databufs is the number of 512 Seed records that can be stored in the comserv and defaults to 20. Detbufs, timbufs, calbufs, msgbufs, and blkbufs are the number of Seed records of the appropriate type that can be stored in comserv and they all default to 20.

Netto is the number of seconds allowed to get any kind of valid packet from the DA before a TCP network connection will be closed. The default is 120 seconds and does not apply to serial or UDP connections. Netdly is the delay in seconds between checking for a new TCP connection. The default is 30 seconds and does not apply to serial or UDP connections.

Reconfig is the number of sequence errors that are tolerated before a link reconfiguration is done, the default is 25. Grpsize specifies how many packets from Comserv to the DA need to be available to send before the whole group is sent. The default grpsize is 1, indicating that there is no grouping, packets are sent as soon as they are available. Grptime is a timeout for a group, so that available packets are sent in a timely manner even if the whole group is not available, defaults to 5 seconds.

There are four types of clients : Transient, Reserved, Blocking, and Foreign. Transient clients are not identified in the configuration file and they do not occupy specific places in the client table or in service queues. Their names need not be unique (you could have multiple clients name "TEST").

Reserved clients have their names identified in the configuration file without the optional timeout value. They have a reserved slot in the client table and service queue and therefore have higher service priority than transient clients, they must also have unqiue names.

Like reserved clients, blocking clients are identified in the configuration file and in addition have a timeout specified in seconds. Blocking clients may actually die and come back without losing data or having data that has already been acknowledged being sent to them again. In addition, if a blocking client does not acknowledge packets it is possible that comserv will have to "suspend" link activity. In this instance, packets will backup in the DA memory and therefore it is not useful to specify a timeout that is longer than the capability of the DA to buffer. Should a blocking client not make a service call for the duration of the timeout period, comserv will mark that client as inactive and free up any packets that are waiting for that client. In the definition "clientn" N can be any string, the client table is filled in the order of declaration, the first declaration having the highest priority when it comes to processing service requests from clients. Blocking clients are assumed to have done an implicit "attach" call to the server when the server starts, this allows blocking clients to start after the servers and not lose data (as long as they start within the timeout value).

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

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

Other commands are priviliged and are only available to foreign clients if there is a "uidnn=mask" entry in the server configuration file where "nn" is the foreign client's user ID. Mask is the sum (decimal) of the following values depending on which commands are to be enabled :

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

If the DA is a SHEAR system rather than an Ultra-SHEAR system, additional information must be provided :

```
[shear]
network=xx
seqmod=sequence-modulus
calib=SUPERCAL | QAPCAL
stream-name=[LL-]SSS[#n],[LL-]SSS[#n]....
      .
stream-name=..[LL-]SSS[#n],[LL-]SSS[#n]..
```

Network is the Seed Network identifier (1 or 2 characters). Seqmod is 8 if not specified but might be less for some special version of Quanterra DA software (Berkeley for example).

Calib is optional and specifies that up to 2 SUPERCAL or QAPCAL calibrators are installed and can be remotely operated using the old style commands.

Up to 7 stream specifications can be made. These allow mapping component/stream combinations to SEED names. If not specified the data is still available, but default SEED names will be used. As an example, suppose that in the Aqcfg file for the DA the component list is :

```
z n e zm nm em it p
```

Assuming that z, n, and e are sent as VBB, VSP, and LP data, and zm, nm, and em are sent as VLP data, and it and p are sent as ULP data :

```
vbb=bhz#1,bhn#2,bhe#3
vsp=ehz,ehn,ehe
lp=lhz,lhn,lhe
vlp=,,,vmz,vmn,vme
ulp=,,,,,,uki,uep
```

Would define SEED names for those channels. Seed locations can also be specified, such as "AB-BHZ". The optional # followed by a number is the physical digitizer channel number (1-n). This information is used by clients wishing to correlate SEED names with calibrator boards. This information is only relavant to channels that have calibrators and need only be listed for one of the streams.

DA Configuration Notes

1) For Ultra-Shear and Multi-Shear DA's you should set "rce=y" in the [comlink] section of the config file. This allows clients to know that the DA actually received the command, rather than just assuming it did.

Programs Supplied

The following programs (including path name and command line options) are provided in this release :

`comserv/comserv station`

This is a server program, one invocation is required for each DA. "Station" is the name of the station that this server will talk to and is used as an index into the previously described master station file. Normally all servers will be started before any clients, but this is not required.

`clients/dataread [station] [-v]`

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

`clients/msgmon station`

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

`clients/config [station]`

This program goes through all stations (or the specified station) showing configuration information available at that time about the stations. It is useful if you wan't to save a "reference configuration" for a station on hard copy or disk for later referal. It's client name is "CONF".

`clients/dpda station`

This is an interactive client used to excercise all available commands available from comserv (excluding reading data records and blockettes, see dataread). It certainly is not a production program but does show how to set up the data structures and excecute the commands. It's client name is "CMDS".

Interface Library and Include Files

The main files you need to look at are :

include/dpstruc.h

This file includes definitions for all commands and responses available from comserv. These structures will be described later.

include/service.h

This file has definitions for actually accessing comserv, comserv structures known to clients, and client structures known to comserv. It also contains prototypes for comserv access library routines :

cs_setup (address of tstations_struc, address of clients name, address of server's name, shared command buffer flag, blocking flag, buffer count, selector count, data mask, command output buffer size)

Tstations_struc is a temporary structure that is filled in by cs_setup and used by cs_gen, after that it has no use. You can either use a variable for this or create it dynamically before calling cs_setup and then destroy after calling cs_gen. cs_setup initializes this structure based on the defaults given in it's parameter list. You can change this structure individually for each station if you wish before calling cs_gen.

The client name is a null terminated string of up to 4 characters long and should inidicate the general function of the client if possible. The server name is a null terminated string of up to 4 characters long or an asterisk indicating you want to setup communications with all servers in the master configuration file.

If the shared command buffer flag is true then there will only be one command input and output buffer that will be shared among all servers. This is the normal case unless you want to get fancy and overlap commands between stations in which case set this flag to false.

The blocking flag if true, indicates that you want to be sure that you don't miss any data while you are gone doing something else (such as a Tape handler client). This flag is ignored by comserv unless you are listing in the configuration file for that station as being a blocking client.

Buffer count is the maximum number of data records or blockettes you wish to be able to receive in any one service call to comserv, 10 is a good starting value. Selector count is the maximum number of Seed name/location selectors you wish to be able to use. Each selector is 6 byte string in the format "LLSSSx" where LL are the location, SSS are the Seed Channel ID, and x is a don't care (normally 0 for C clients). Question marks are used as "don't cares". You must specify at least one selector which cs_setup will initialize to "?????" and use for all data and blockette requests. If you wish to receive only selected data, or to be able to request detector information then you need at least 2 selectors.

Data mask is any combination of :

CSCM_DATA	1	512 Mini Seed records
CSCM_EVENT	2	Event Detection Blockettes
CSCM_CAL	4	Calibration Blockettes
CSCM_TIMING	8	Timing Blockettes
CSCM_MSG	$10	Message Blockettes
CSCM_BLK	$20	General Blockettes, such as GPS

and describes the data you will want to receive.

Command output buffer size must be at least 100 and may need to be fairly large for a complex DA. The example programs use 6000 but that should be considered a maximum required. If a comserv command detects too small of a buffer it will return with the CSCR_SIZE status.

After execution the cs_setup function will have setup the tstation_struc using the defaults given. The (tstation_struc) starts with the header (Field and structure names given in parenthesis) :

Client name (myname)		
(shared) Flag	0	Number of stations (station_count)
Number of data buffers (data_buffers)		0
First station through last stations (station_list)		

The (station_list) is defined as (tstation_entry) :

Station's name (stationname)		
Command output size (comoutsize)		
(selectors) count		Data (mask)
(blocking) flag	0	0
Segment Key for this station (segkey)		
(directory) for station, char119		

If you do not have shared command output buffers, you could edit the (station_list) entries to customize the size of each buffer. You could also customize (selectors), (mask), and (blocking) for each station. If you wish to remove a station from the list you can use :

cs_remove (address of tstations_struc, station_number)

Where station_number is between 0 and (station_count).

After doing any customizations of the the (tstations_struc), you call cs_gen to actually generate the shared memory segment for your client :

cs_gen (address of tstation_struc) returns address of (tclient_struc)

This function is passed the (tstation_struc) address and returns you with the address of your shared memory segment, which is defined as (tclient_struc) :

Client's Name (myname)		
Client's Process ID (client_pid)		
Client's Shared memory ID (client_shm]		
Client's User ID (client_uid]		
Service is (done)	(spare) 0	Service (error) code
(maxstation) available		(curstation) for service
(offsets) from module to each (tclient_station) structure		

Notice that the location of each (tclient_station) structure is given as an offset from the beginning of the shared memory segment. This is because Unix does not guarantee that each process will see the memory segment at the same address (unlike data modules used on OS-9). The offset table can be addressed from 0 to (maxstation) -1 to get the offset for the desired station. Adding this to the beginning address of the shared memory segment will give you a pointer to a (tclient_station) :

Station (name)		
Station segment key (seg_key)		
(command) to service	(blocking) service	station (status)
Sequence for (next_data) packet		
Time at (last_attempt) for service		
Time of (last_good) service call		
Server reference code (servcode)		
Address of Comserv's shared memory segment (base)		
Offset to Command Input Buffer (cominoffset)		
Offset to Command Output Buffer (comoutoffset)		
Size of Command output buffer (comoutsize)		
Offset to Data Buffer Array (dbufoffset)		
Size of Each Data Buffer (dbufsize)		
Number of buffers (maxdbuf)	Requested buffers (reqdbuf)	
Valid buffers after call (valdbuf)	Buffer sequence control (seqdbuf)	
Data starting time (startdbuf)		
Offset to Selector Array (seloffset)		
Number of Selectors (maxsel)	0	
Selector ranges (sels) [0..6]		
Data (mask)	0	
Buffers follow		

(command) is a constant of the form "CSCM_xxxxx" which will be described later. The station (status) has one of the following values :

CSCR_GOOD	0	No problemo
CSCR_ENQUEUE	1	Could not find a service queue slot to put my request in
CSCR_TIMEOUT	2	Server did not process my request within the timeout.
CSCR_INIT	3	Server memory segment still in initialization.
CSCR_REFUSE	4	The Server is already handling the maximum number of clients.
CSCR_NODATA	5	The information requested is not currently available.
CSCR_BUSY	6	The command requested requires resources not currently available.
CSCR_INVALID	7	The information requested cannot be available due to comlink mode used.
CSCR_DIED	8	The server has died, try again later.
CSCR_CHANGE	9	The server has re-appeared.
CSCR_PRIVATE	10	Could not attach to the memory segment given for an upload.
CSCR_SIZE	11	Your command output buffer is not large enough to process your request.
CSCR_PRIVILEGE	12	Foreign client cannot execute this command.

The sequence for the (next_packet) is used by the server and client service libraries to determine whether the server has new data or blockettes for you. You should not manipulate this value directly, use (seqdbuf) instead.

Time values are 8 byte double precision times in seconds since 1970. If your client needs the seconds since 1984 then simply subtract 441763200 (this is defined in dpstruc.h as SECCOR).

Time at (last_attempt) for service is the last time the client actually did a service request, while Time of (last_good) service call is the time at which the server completed a service request for this client. These values are used by the library routines to periodically poll a dead server to see if it has come back. Server reference code (servcode) is the time at which the server started up and is used to determine that a new incarnation of the server has started to make the appropriate data structure changes.

Address of Comserv's shared memory segment (base) is used by library routines to make the actual connections to the server, it points to (tserver_struc).

Offsets to the Command input buffer (cominoffset), Command output buffer (comoutoffset), Data buffer array (dbufoffset), and Selector Array (seloffset) are again, values to be added to the beginning of the memory segment of where those structures start.

The Command input buffer definition is completely determined by the command being requested. The command output buffer has a standard header (comstat_rec) followed by command specific fields :

(command_tag), 1-255 command sequence number
(completion_status) Command buffer status
Command specific information (moreinfo)

Many commands (those that return information to the client) assign a new command sequence number and put it into the comout buffer as (command_tag). Do not modify this value. The completion status indicates if the data is ready for the client to process, the valid values are :

CSCS_IDLE	0	No command being processed
CSCS_INPROGRESS	20	Server is working on it
CSCS_FINISHED	21	Server has completed the request, data available.
CSCS_REJECTED	22	Not currently used
CSCS_ABORTED	23	File upload/download aborted by client
CSCS_NOTFOUND	24	File requested for download not found on DA
CSCS_TOOBIG	25	File is too big to transfer (> 65000 bytes)
CSCS_CANT	26	DA cannot create upload file requested.

The data buffer structure is defined as an array [0 .. Number of Data Buffers(maxdbuf)] of (tdata_user) :

(reception_time) of packet
(header_time) of packet
(data_bytes) 512 Byte Mini-Seed Record or one (shorter) blockette

The Selector array structure is defined as an array[0 .. Number of Selectors(maxsel)] of array[0..5] of char.

Selector ranges (sels) have the (selrange) type :

(first) selector	(last) selector

There are 4 of these entries, each for a different function. The array index for the selector ranges are :

DATAQ	0	Selectors for 512 byte mini-seed records
DETQ	1	Selectors for detection blockettes
CALQ	2	Selectors for calibration blockettes
TIMQ	3	Selectors for timing blockettes
MSGQ	4	Selectors for message blockettes.
BLKQ	5	Selectors for general blockettes
CHAN	6	Selectors for requesting channel information or detectors.

cs_gen initializes the first (0)selector to ????? and all selector ranges to 0-0. Selector ranges for each queue are completely independent of each other.

Buffer Sequence control (seqdbuf) has the following four values :

CSQ_NEXT	0	Get data newer than I already have
CSQ_FIRST	1	Get first available data
CSQ_LAST	2	Get new data only

When a client starts or if the server restarts (seqdbuf) is set to CSQ_FIRST by the libraries. After each data request the server sets (seqdbuf) to CSQ_NEXT. You may manually set (seqdbuf) to CSQ_LAST if you need to. (startdbuf) is the earliest packet that will be returned (regardless of type). (startdbuf) is initialized to 0.0 (1970) by cs_gen, but you can change this field if you want to skip over earlier data..

The data (mask) is initialized to the value in the (tstation_struc) but can be changed at any time.

cs_svc (address of (tclient_struc), station_number) returns station status

This is the basic service request to comserv. The first parameter is the pointer returned by the cs_gen call. The second parameter is the station number that you want this command to be sent to. It must be in the range of 0 to (maxstation) - 1. This command can be used to send any command, but is generally not used to send the data request command (CSCM_DATA_BLK). If the command returns data then it will either set the completion status to CSCS_INPROGRESS if it cannot immediately provide the data, or CSCS_FINISHED when it has the data. Before executing another command you need to set the status to CSCS_IDLE before executing another command. If the completion status was CSCS_FINISHED, then you only need to do a simple assignment into the completion status field. If the completion status was CSCS_INPROGRESS, you will need to issue a CSCM_CMD_ACK command. Refer to the example program "dpda" for illustrations of the proper procedure.

cs_scan (address of (tclient_struc), address of alert flag) returns station number or -1 (NOCLIENT)

This command is normally to handle getting data records and blockettes from multiple stations. It returns the station number of the first station it finds (it uses a round-robin search) that has data available or if the station status changes. If the alert flag (boolean) is set to TRUE, then check the station (status) field. Data and blockettes are available if the (valdbuf) field is non-zero. This command uses the cs_svc, cs_check, cs_link, and cs_attach calls.

cs_off (address of (tclient_struc))

This command should be called before a client terminates to clean up shared memory segments.

The following commands probably don't need to be called by clients (none of the examples clients do), but are available if required.

cs_link (address of (tclient_struc), station number, FALSE)

Tries to link to the server's shared memory segment for the indicated station. If that works, it verifies that the server PID is still alive. Sets the station (status) appropriately.

cs_attach (address of (tclient_struc), station number)

Tries to send the CSCM_ATTACH command to the server to verify proper operation. Sets the station (status) appropriately.

cs_check (address of (tclient_struc), station number, current time) returns status or indication of action

This routine is the real guts of the cs_scan call above. This routine goes through the following steps :

1) If it does not have good status, then every 10 seconds it tries :
   a) cs_link, and if good, does :
   b) cs_attach.
2) If a station has good status then it checks the server reference code to make sure the server hasn't changed invocations while the client was away. If it did it :
   a) Resets the next record counters and changes the sequencing to CSQ_FIRST.
   b) Sets the status to CSCR_CHANGE and returns the same.

3) If the station has good status and the server has not changed then it checks to see if the server has newer data or blockettes than the ones the clients has. If so it returns with CSCR_GOOD, if not returns CSCR_NODATA. To keep the link alive it will return with CSCR_GOOD if no activity within 10 seconds to force a service request by cs_scan.

The server shared memory segment (at least the part accessable by the client's library routines) is defined as (tserver_struc) :

(init) character	0	0
Server's Process ID (server_pid)		
Server's Memory ID (server_semid)		
Server's User ID (server_uid)		
Number of microseconds client shoud wait for server (client_wait)		
Microseconds per wait for non-foreign clients (privusec)		
Microseconds per wait for foreign clients (nonusec)		
Packet number for next data to be received (next_data)		
Server reference code (servcode)		
Array of service queues [0..MAXCLIENTS]		

The (init) character is 'I' if the server has finished initialization of it's shared memory segment. The client should not attempt access unless this is 'I'. (client_wait) is used by client library routines to determine when it should give up waiting for a response (this is currently 10 seconds). (privusec) is the number of microseconds for each wait call for non-foreign clients and defaults to 1 second, while (nonusec) is for foreign clients, defaults to 100ms. The packet number is used by client library routines (cs_check) to determine whether it is likely that the server would have new data for the client should data be requested. The service array queue has the following format on Unix systems :

Client's shared memory segment (clientseg), -1 (NOCLIENT) if empty
Client's name (reserved clients only)

cs_svc is the routine that actually puts the clients memory ID into the server service queue. It can use a slot with a client name in it only if it matches the client's name, else it can use any slot that has no client name (not reserved).

include/seedstrc.h

This file contains SEED definitions used by comserv. The client is not required to use these definitions, they are provided for your convenience.

include/stuff.h

This file contains definitions for routines in util/stuff.c that you might find useful in clients :

dtime () returns double precision seconds since 1970

General routine to find current time.

str_long (pointer to C string) returns long integer

Converts a null terminated string into a long integer, if the string is shorter than 4 characters then it is padded with spaces the on right. Station and client names are stored and used as long integers to speed execution.

long_str (longinteger name) returns pointer to C string

Similar to str_long, but converts the other way around.

strpcopy (pointer to room for C string, pointer to Pascal string)

Converts the Pascal string to a C string.

strpas (poitner to room for Pascal stirng, pointer to C string)

Converts the C string to a Pascal string.

set_bit (pointer to long integer, bit number)

Sets the indicated bit number (0 - 31) into the long integer bit mask.

clr_bit (pointer to long integer, bit number)

Clears the indicated bit number (0 - 31) from the long integer bit mask.

test_bit (long integer, bit number) returns TRUE or FALSE boolean value

If the indicated bit number (0 - 31) is set in the mask, then return TRUE, else return FALSE.

untrail (pointer to C string)

Removes any trailing spaces from the string.

upshift (pointer to C string)

Converts any lower case letters in the string to upper case.

addslash (pointer to C string)

If the string does not end in a '/', adds one.

str_right (pointer to room for C string, pointer to middle of C string)

Starts at the character after the middle pointer and copies characters to the first parameter string until the null terminator.

longhex (unsigned char value) returns long integer

Converts the 8 bit value to a 32 bit value, with zero filling (no sign extension).

ord (unsigned char value) reutrns short integer

Converts the 8 bit value ot a 16 bit value, with zero filling (no sign extension).

include/timeutil.h

This module has many routines, most of which are of use only to comserv, however, this module defines those that might be of interest to clients. They are :

lead (final character count, padding character, pointer to C string) returns pointer to C string

Adds leading characters to a string until the string is the specified length.

time_string (double precision seconds since 1970) returns pointer to a C string

Converts seconds since 1970 to a C string.

localtime_string (double precision seconds since 1970] returns pointer to a C string

Converts seconds since 1970 to a C string, but in local time instead of UTC.

jconv (year, julian day) returns long integer seconds since 1970

Useful for converting SEED times (which use a strange combination of gregorian and julian fields) to seconds since 1970.

COMSERV COMMANDS

The following commands are available :

CSCM_ATTACH

This command is essentially a "NOP" and basically just lets the server know you are still there and lets the client know if the server is still there. There are no parameters.

CSCM_DATA_BLK

This command is used to request new data from the server and is generally only issued by the cs_scan library routine. In (tclient_struc) the following fields must be valid before making this call : next_data, reqdbuf, seqdbuf, sels, and mask.

CSCM_LINK

The following will be returned in the Command Output Buffer (link_record) :

(window_size) in packets	(total_prio) total priors.	(total_prio) total priors.	(det_prio) detection prio.
(time_prio) timing prio.	(cal_prio) calibration pr.	(link_format)	(rcecho) Command echo
(resendtime) Packet resend timeout		(synctime) Sync Packet Interval	
(resendpkts) Number of packets resent at a time		(netdelay) Network restart delay	
(nettime) Network connect timeout		(netmax) Unacked network packet before timeout	
(groupsize) Group packet count		(grouptime) Group timeout	

Link Format is one of the following :

CSF_QSL	0
CSF_Q512	1

CSCM_CAL

The following is returned in the Command Output Buffer, first the header (cal_record) :

(number) of calibrators	(mass_ok)	0

(mass_ok) is TRUE if any of the calibrators support mass recentering. The following block is repeated up to 4 times and follows the above header. It can be accessed as part of the above structure as (acal[n]) where n is between 0 and (number) - 1. It has the type (eachcal).

(coupling_option) avail.	(polarity_option) avail.	(board) number 1-n
(min_settle) time in seconds		(max_settle) time in seconds
(inc_settle) time in seconds		(min_mass_dur) in milliseconds
(max_mass_dur) in milliseconds		(inc_mass_dur) in milliseconds
(def_mass_dur) in milliseconds		(min_filter) number, 0 if no filters
(max_filter) number, 0 if no filters		(min_amp) in dB
(max_amp) in dB		(amp_step) in dB
(monitor) channel, 0 if none		(rand_min_period) 0 if none
(rand_max_period) 0 if none		(default_step_filt) 0 if none
(default_rand_filt) 0 if none		(ct_sp2) 0
(durations) limits for each waveform, in seconds, array of (tdurations)		
(map) of physical digitizer channels this calibrator supports		
Supported (waveforms) bit map (Sine, Step, Red Noise, White Noise)		
Supported (sine_freqs) bit map, see "HZxxxx" bit constants in dpstruc		
(default_sine_filt) array 0..MAXCAL_FILT-1 of long integer bit maps		

Calibrator Board (name) - String23
(filtf) Calibrator Filter description - String59

(tdurations) is defined as :

(min_dur) minimum duration in seconds
(max_dur) maximum duration in seconds
(inc_dur) duration increment in seconds

The waveform types are defined as

SINE	0	Sine wave
STEP	1	Single step
RAND	2	Red Noise
WRAND	3	White Noise

CSCM_DIGI

The following is returned in the Command Output Buffer (digi_record) :

Digitizer (name) - String23			
Digitizer (version) - String23			
(clockmsg) Prompt - String79			
(prefilter_ok) flag	(detector_load_ok) flag	(setmap_ok) flag	(clockstring_ok) flag
(int_ext_ok) flag	(send_message_ok) flag	(message_chan_ok) flag	(set_osc_ok) flag
(set_clock_ok) flag	(wait_for_data) Time	(dt_sp1) 0	(dt_sp2) 0

CSCM_CHAN

Information for the requested channels is returned in the Command Output Buffer. The channels described will be those defined by sels[CHAN]. The buffer starts with the header (chan_struc) :

(chancount) number of channels	Array of channels (chans) 0 ... (chancount) - 1

Each entry has the following format (chan_record) :

(seedname) Seed Channel ID			(stream)
(seedloc) Seed Location		(physical) digi. channel	(available) Mask
(enabled) Mask	(det_count) # of detectors	(c_prio) Cont. Priority	(e_prio) Event Priority
Sampling (rate) + = Samples/Sec, - = Seconds/Sample			

The Available and Enabled masks have the following format :

- Bit 0 = Continuous data on this Comlink
- Bit 1 = Event data on this Comlink
- Bit 2 = Continuous data on Tape
- Bit 3 = Event data on Tape
- Bit 4 = Continuous data on Disk
- Bit 5 = Event data on Disk

CSCM_ULTRA

The following information is returned in the Command Output Buffer (ultra_rec) :

(vcovalue) 0 - 4095		(pllon) flag	(umass_ok) flag
Comm-Event mask (comm_mask)			
(ultra_rev) Revision level	First Comm Event Name - Variable length Pascal Strings, 32 names total.		

The Comm Event Name strings follow each other with no breaks (after the last character of a string immediately follows the dynamic length of the next).

CSCM_LINKSTAT

The following information is returned in the Command Output Buffer (linkstat_rec) :

(ultraon) flag	(linkrecv) flag	(ultrarecv) flag	(suspended) flag
(total_packets) Total packets received			
(sync_packets) Sync packets			
(seq_errors) Sequence errors			
(check_errors) Checksum errors			
(io_errors) I/O Errors			
(lastio_error) Last I/O Error			
(blocked_packets) Number of blocked packets			
(seconds_inop) Seconds server has been in operation			
(last_good) Time of last good packet received - 8 byte double precision			
(last_bad) Time of last bad packet received - 8 byte double precision			
(seedformat) char[4], currently V2.3 - not null terminated			
(seedext) Extensions - 'B'	(data_format) CSF_QSL	(description) of station string59	
(pollusecs) Server polling delay in microseconds			
(reconcnt) Sequence errors before reconfiguring link			
(net_idle_to) Timeout before disconnecting network if no packets received from DA			
(net_conn_dly) Network connection polling delay			
(grpsize) Grouping size for DP to DA packets			
(grptime) Timeout for grouping			

CSCM_UNBLOCK

This command is used to unblock packets for a client and to disable that client from blocking any more packets until that client returns. The command input buffer is simply the client number (0 - n) :

Client Number

CSCM_RECONFIGURE

This command tells the server to reconfigure (reread link status and ultra information from DA). There are no parameters.

CSCM_SUSPEND

The server stops acknowledging packets from the DA, this command is normally a prelude to shutting down a server. CSCM_LINKSTAT can be used to determine when all blocked packets have been read, and therefore safe to terminate the server.

CSCM_RESUME

This reverses the action of CSCM_SUSPEND.

CSCM_CMD_ACK

Clears out the client's command status to CSCS_IDLE and releases the client command pointer inside comserv, allowing it to process other requests. The server normally clears it's own pointer when it completes a command, but this command is provided should this not happen for some reason.

CSCM_TERMINATE

Terminates the server.

CSCM_LINKSET

Modifies the link parameters for Comserv.

(pollusecs) Server polling delay in microseconds
(reconcnt) Number of sequence errors before reconfiguring link.
(net_idle_to) Disconnects network and reconnects if no packets received for this time
(net_conn_dly) Number of seconds to wait between checking for network connections.
(grpsize) Size of DP to DA packet grouping.
(grptime) Timeout for grouping, packets sent after this timeout even if group not full.

CSCM_SHELL

Instructs the DA to execute the OS9 shell with supplied parameter, having it's standard output redirected to a pipe so that it can be returned. The command input buffer must be set to (shell_com) :

(shell_parameter) String79	
(log_local) 0	(log_host) 0

Results of executing this command will be shown as messages from the DA. If an Ultra-Shear DA has remote command echo **(RCE)** enabled then the status of this command will be CSCS_INPROGRESS until the echo is received, at which point it will change to CSCS_FINISHED. If using a Shear DA or remote command echo is disabled then CSCS_FINISHED will be returned immediately.

CSCM_VCO

Instructs the DA to enable/disable the Phase lock, and if disabled, to manually set a new VCO control value. RCE is applicable to this command. For Shear systems comserv will translate this into a command that can only set the VCO value modulus 16. The command input buffer must be set to :

New VCO Value - $FFFF=PLL Control

CSCM_LINKADJ

Instructs the DA to set new Commo parameters. The command input buffer must be set to (linkadj_com) :

(window_size) Size of DA to DP window in packets		(set_msg) Msg. Priority	(set_det) Det. Priority
(set_time) Timing Prior.	(set_cal) Cal. Priority	(resendtime) Timeout before resending packets	
(synctime) Sync packet interval		(resendpkts) Number of packets resent at a time	
(netdelay) Network reconnection delay		(nettime) Network connection timeout	
(netmax) Number of resent packets before net. discon.		(groupsize) Number of packets to group together	
(grouptime) Groupt timeout		(lasp1) Spare number 1 - set to zero	
(lasp2) Spare number 2 - set to zero			

CSCM_MASS_RECENTER

Instructs the DA to send a mass recentering command to the specified calibrator board. RCE is applicable to this command. For Shear systems, comserv will translate this into an old command which does not allow selection of board or duration. The command input buffer must be set to (recenter_com) :

(board) number	(duration) in milliseconds

CSCM_CAL_START

Instructs the DA to start a calibration command on the indicated board. RCE is applicable to this command. For Shear systems, comserv will translate this into an old command which does not allow selection of settling time or filter number. The command buffer must be set to (cal_start_com) :

(calcmd) Waveform	(sfrq) sine frequency	(plus) Step	(capacitor) Coupling
(autoflag) GTSN use	(ext_sp1) 0	(calnum) Board number	
(duration) in seconds			
(amp) in dB		(rmult) random step multiplier	
Channel (map) relative to this board		Relay (settle) time in Seconds	
(filt) Number		(ext_sp2) 0	

CSCM_CAL_ABORT

Instructs the DA to stop a calibration command on the indicated board. RCE is applicable to this command. For Shear systems comvserv will translate this into an old command. The command buffer must be set to :

Board Number

CSCM_DET_ENABLE

Instructs the DA to turn on or off up to 20 detectors. RCE is applicable to this command. The Command input buffer first has the count of detectors to enable/disable (det_enable_com) :

(count) 1-20	Array of (detectors) 0 .. 19 of (det_en_entry)

The array contains up to 20 of the following blocks (det_en_entry) :

(detector_id)	(enable) Flag	(de_sp1) 0

CSCM_DET_CHANGE

Allows you to change the parameters of one detector in the DA. RCE is applicable to this command. The command input buffer must be set to (det_change_com) :

Detector (id)		(enab) flag	(dct_sp) 0
Detector parameters (ucon) data type is (shortdetload)			

(shortdetload) is defined as this 42 byte structure :

(filhi)
(fillo)
(iwin)
(n_hits)
(xth1)
(xth2)
(xth3)
(xthx)
(def_tc)
(wait_blk)
(val_avg)

CSCM_REC_ENABLE

Allows setting recording flags on up to 8 channels on the DA. RCE is applicable to this command. The command input buffer starts with the count of channels to change (rec_enable_com) :

(count) of channels	Array (changes) 0 .. 7 of data type (rec_one)

Followed by up to 8 of the following blocks (rec_one) :

(seedname) Seed Channel ID		Enable (mask)
(seedloc) Seed Location	(c_prio) Cont. Priority	(e_prio) Event Priority
(rec_sp1) 0		

CSCM_COMM_EVENT

Sets the remote detector bit map in the DA. RCE is applicable to this command. The command input buffer must be set to :

Remote Detector Bit map (remote_map) 1=on, 0=off, per bit
Remote Detector Bit mask (remote_mask) 1=change based on bit map, 0=no change, per bit.

CSCM_DET_REQUEST

Used to obtain parameters on all detectors that are available for a given DA channel (up to 20). RCE is not applicable to this command because the data must be received from the DA, therefore this command will always return with command status set to CSCS_INPROGRESS. (sels[CHAN]) must specify only one specification, this is the channel for which the following data will be returned. When the command has completed, the Command Output Buffer will contain the information requested, first the header (det_request_rec) :

(count) of detectors	(dat_sp1) 0
Array (dets) 0 .. 19 of structure (det_descr)	

And then up to 20 of the following blocks (det_descr) :

(enabled) flag	(remote) flag	(dettype) 0/1	(dd_sp1) 0
(cons) contains the (shortdetload) structure - 42 bytes		Detector (id)	
Detector (name) String23			
Detector Type Name - String15 (params[0])			
Filhi Description - String15 (params[1])			
Fillo Description - String15 (params[2])			
Iwin Description - String15 (params[3])			
Nhits Description - String15 (params[4])			
Xth1 Description - String15 (params[5])			
Xth2 Description - String15 (params[6])			
Xth3 Description - String15 (params[7])			
Xthx Description - String15 (paams[8])			
Def_tc Description - String15 (params[9])			
Wait_blk Description - String15 (params[10])			
Val_avg Description - String15 (params[11])			

CSCM_DOWNLOAD

Requests that the specified file on the DA be downloaded and placed in the shared memory segment RCE is not applicable to this command. The command buffer must be set to (download_com) :

(dasource) file name - String59
(dpmodname) Not Used - string23

The command output buffer will have the following data structure (download_result) :

(dpshmid) Shared memory segment ID	
(fsize) file size in bytes - unsigned	(byte_count) received so far - unsigned

When file transfer is complete status will change to CSCS_FINISHED. You should delete the shared memory segment (on Unix) after you have made the file. Note that on Unix systems you need to convert CR to LF in text files when writing to disk.

CSCM_DOWNLOAD_ABORT

Requests Comserv to abort the download, there are no parameters.

CSCM_UPLOAD

Requests that the data in the shared memory segment be sent to the DA. Note that on Unix you need to convert LF to CR for text files. The command buffer must be set to (upload_com) :

(dadest) File name on DA - string59
(dpmodname) Not Used - string23
(dpshmid) shared memory segment ID
(fsize) file size in bytes - unsigned

The command output buffer will have the following status record (upload_result) :

(bytecount) transferred so far - unsigned	(retries) packets resent

A word explanation is probably in order here. The algorithm sends all packets first, incrementing the (bytecount) with the number of bytes sent as each packet is sent. After all packets are sent it polls the DA to determine which packets it actually received. It then starts resending selected packets, incrementing the (retries) counter, once per packet. The polling and resending continues until all packets are acknowledged by the DA. At this point status will change to CSCS_FINISHED and comserv should have deleted the shared memory segment.

CSCM_UPLOAD_ABORT

Requests that the upload in process be aborted, there are no parameters.

ADDITIONAL NOTES

If it is desired to shut down the Comserv process, then a utility program should be available to send the CSCM_SUSPEND command. The process should then periodically poll using the CSCM_LINKSTAT command to determine how many packets have still not been read by all blocking clients (or some maximum of time), and then kill the Comserv process using the CSCM_TERMINATE command. This capability is provided so that Comserv does not get killed while it still has packets that it has told the DA it received, but has not delivered to all blocking clients.

If using a network connection, it is recommended that you use ftp to transfer files rather than the built in upload and download. In particular upload (file from host to DA) has some unresolved timing problems when running on a network connection. FTP is faster in any case.

OS9 NOTES

There is a file called "os9stuff.h" that defines various functions to simulate shared memory segments and semaphores under OS-9. Note that these routines are specifically written to perform just those functions required for comserv and clients, they are not a general purpose simulation.

```
/* return memory id. If key is IPC_PRIVATE than a unique number
   is generate, else the key is used. A data module will be
   created if shmflag has IPC_CREAT set, else it is simply
   linked to. A data module is used to simulate System V shared
   memory with a name of "SHMxxxxx" where xxxxx is the value of
   the key */
int shmget (int key, int size, int shmflag) ;

/* based on memory id, return data address of the data module.
   This address is the module address + $34. shmaddr and
   shmflag are ignored */
char *shmat (int shmid, char *shmaddr, int shmflag) ;

/* based on the data address, unlink from the data module. The
   module address is the data address - $34. */
int shmdt (char *shmaddr) ;

/* based on the memory id, unload the data module from memory.
   cmd and buf are ignored. */
int shmctl (int shmid, int cmd, char *buf) ;

/* if IPC_CREAT is set in semflag then create new event, else
   link to existing event. The event has a name of "SEMxxxxx"
   where xxxxx is the value of the key. nsems is ignored. */
int semget (int key, int nsems, int semflag) ;

/* opsptr is a point to array of 3 values, if the middle value
   is +1 then this is a ev_signl, if it is -1 then this is a
   ev_wait. nops is ignored. */
int semop (int semid, struct sembuf *opsptr, unsigned int nops) ;

/* compare two strings, ignoring case. For some reason, Microware forgot this one */
int strcasecmp(const char *string1, const char *string2) ;

/* compare two strings, ignoring case, for a maximum of "n" characters */
int strncasecmp(const char *string1, const char *string2, int n) ;

/* change size of interrupt input buffer, returns error code */
int resetp (int path, int size) ;

/* Enable hardware flow control, returns error code */
int hardon (int path) ;

/* Try to a block read from the path. returns error code */
int blockread (int path, int bytecnt, pchar buf) ;
```

OS9 Pascal Interface

On OS9 systems, clients can be written in OmegaSoft Pascal if desired. There are two include files, one is comservstrc.i and is similar to a combination of dpstruc.h and service.h. Note that in this release, not all data structures have been converted to Pascal. This applies mostly to comserv specific structures, basic data structures can be found in various existing include files. Comservice.i has the definitions of procedure and functions to communicate with comserv :

```
{
  This procedure will setup the tstations_struc for all comlink stations
  found in the "/r0/stations.ini" file or the specified station. You can then
  customize stations to suit your needs if you desire. For each station it puts
  in the default number of data buffers and selectors specified. If shared is TRUE,
  then there is one shared command input/output buffer for all stations (of size
  comsize), else each station has it's own buffer. If blocking is TRUE, then
  a blocking connection will be requested.
}
  procedure cs_setup (var stations : tstations_struc ; name, sname : namestring ;
      shared, blocking : boolean ; databufs, sels, mask : integer ;
      comsize : longinteger) ; external ;

{
   Remove a selected station from the tstations_struc structure, must be
   called before cs_gen to have any effect.
}
  procedure cs_remove (var stations : tstations_struc ; num : integer) ; external ;

{
  This function takes your tstations_struc and builds your shared memory segment
  and returns it's address. This should be done when client starts. You can
  either setup the tstations_struc yourself, or use the cs_all procedure.
  It attaches to all servers that it can.
}
  function cs_gen (var stations : tstations_struc) : pclient_struc ; external ;

{
  This function detaches from all stations and then removes your shared
  memory segment. This should be called before the client exits.
}
  procedure cs_off (client : pclient_struc) ; external ;

{
  This is the basic access to the server to handle any command. The command
  must have already been setup in the "tclient_station" structure and the
  station number into "curstation" in the "tclient_struc" structure before
  calling. Returns one of the "CSCR_xxxx" values as status.
}
  function cs_svc (client : pclient_struc ; station_number : integer) : integer ; external ;

{ This is a polling routine used by cs_scan. The current time is the third
   parameter. It will check the station to see :
     1) If it does not have good status, then every 10 seconds it tries :
         a) cs_link, and if good, does :
         b) cs_attach.
     2) If a station has good status then it checks the server reference code
        to make sure the server hasn't changed invocations while the client
        was away. If it did it :
         a) Resets the next record counters and changes the sequencing to CSQ_FIRST.
         b) Sets the status to CSCR_CHANGE and returns the same.
     3) If the station has good status and the server has not changed then it
        checks to see if the server has newer data or blockettes than the ones
        the client has. If so it returns with CSCR_GOOD, if not returns CSCR_NODATA ;
        To keep the link alive it will return with CSCR_GOOD if no activity
        within 10 seconds.
}
  function cs_check (client : pclient_struc ; station_number : integer ;
                     now : longreal) : byte ; external ;
```

```
{
  For each station it calls cs_check, and it returns with CSCR_GOOD does a
  CSCM_DATA_BLK command to try to get data. If it gets data then it returns
  the station number, or NOCLIENT if no data. Also, if there is change in
  status of the server (goes away, or comes back for instance) then the station
  number is also returned, and the alert flag is set. You should then check the
  status byte for that station to find out what happened.
}
  function cs_scan (client : pclient_struc ; var alert : boolean) : integer ; external ;

{ try to link to server's shared memory segment. first copies server reference
   code into client's structure }
  procedure cs_link (client : pclient_struc ; station_number : integer ;
                     first : boolean) ; external ;

{ try to send an attach request to the server }
  procedure cs_attach (client : pclient_struc ; station_number : integer) ; external ;

{ Return seconds (and parts of a second) since 1970 }
  function dtime : longreal ; external ;

{ a sleep call that uses a longinteger parameter }
  procedure tsleep (sticks : longinteger) ; external ;

{ convert Pascal string to longinteger }
  function str_long (name : namestring) : longinteger ; external ;

{ Convert longinteger to Pascal string }
  function long_str (name : longinteger) : namestring ; external ;
```