U. S. DEPARTMENT OF THE INTERIOR

U. S. GEOLOGICAL SURVEY

# XDETECT Version 2.01 Technical Reference

by

John A. Rogers

William M. Kohler

Open File Report #OF92-364-A

Menlo Park, California

1992

CONTENTS

**INTENDED AUDIENCE AND SCOPE**

This report is oriented toward the 'C' language programmer who is interested in the details of XDETECT version 2.01 operation. (In addition certain details of XDETECT version 2.03 are also discussed.) This interest might stem from the need to modify or add to the program. Users of XDETECT should also refer to the user's guide (Tottingham, 1990).

**DEVELOPMENT HISTORY**

Computer-based event recording systems have been is use by the U.S. Geological Survey (USGS) since the late 1970s. Up until 1986 this recording was done on mini-computer-based systems costing tens of thousands of dollars. Although low-cost PCs had been around for several years by then, it was still a widely held belief that the PC was not up to the task of network recording. The cost advantage and recent advances in processing power of PC-based systems were so great however, that Willi Lee and John Rogers decided to build a simple 8-channel system to demonstrate this capability. The system was installed in Yakutat, Alaska and acquired seismic data there from 1986 to 1988. This software and its successors gave small networks the benefits of digital data at a price smaller networks could afford.

The Yakutat demonstration system was based on a 4.77 MHz PC/XT with a Data Translation analog input card. It recorded seismic events until 1988 when MDETECT (Tottingham et. al., 1989), based on a PC/AT able to handle 16 channels replaced the PC/XT system. This was followed in 1989 by the 64-channel XDETECT software (Tottingham and Lee, 1989) using an external multiplexer (Ellis, 1990).

Since 1986 many enhancements were made to the original Yakutat hardware and software including the increase of channel capacity from 8 to 128. Some of these improvements were:

1) Real-time graphics - The monitor displays a 512-sample window of the seismogram of the incoming channels as well as program status. The status information given is number is events recorded, begin and end time of the displayed data, free disk space, trigger status, version number and available disk space.

2) Digital Signal Processor (DSP) support - A linkable module controls a 25 MFLOP coprocessor board for real-time spectral analysis and calibration detection (Rogers, 1989).

3) Subnet triggering and recording - A recent version of XDETECT supports flexible triggering and recording in software via user-defined "trigger nets". A trigger net is a group of stations which control a software "record net". A record net is another subset of stations to be recorded which can be set independently of the trigger subnet. There is no software limit on the number of subnets that can be implemented.

4) Continuous data streaming - The incoming data can be recorded continuously.

5) Network support - XDETECT works over networks (see below, XDETECT System Overview). Data can be recorded on network disks as

well as on local disks.

The rapid development of PC hardware and software has increased the computational capability from tens to hundreds of channels while the incorporation of a special DSP co-processor board puts PC power in the super-minicomputer class for a tenth or less the cost.

XDETECT version 2.01 represents several person-years of work and thousands of lines of code. The goal of this report is to provide the framework for a detailed technical reference. Without such a document it will be difficult for software developers to quickly understand the data structures and program flow of XDETECT. This could result in much wasted effort as new recording systems performing the same functions as XDETECT are created. With this reference, however, programmers can concentrate on the more important problems in seismology such as robust trigger or picking algorithms.

The XDETECT program contains 29 modules which are linked together into a single EXE module. As XDETECT evolves some of these modules may change as new features are needed. This report will focus on documenting the most likely-to-be-changed program modules, although most modules will be discussed to some degree. Future revisions of this reference will add more details where needed.

The appendix shows the calling sequence of all the XDETECT functions. It may be most useful if pasted together and hung on a wall.

**XDETECT SYSTEM OVERVIEW**

XDETECT can handle up to 128 channels on an 8 Megahertz PC/AT but a 80386- or 80486-based system is a more typical platform. Data is fed into the PC via a Data Translation DT2824PGH Analog-to-Digital (A-D) converter card. This card can be connected to a 128-channel multiplexer or to a Data Translation DT707 screw terminal board for 16-channel operation. Real-time spectral analysis requires the optional Symmetric Research DSP32 card while network recording is served by either an Artisoft Lantastic v3.11 2 Mbps or a 3Com 3C503 adapter card. For program development work a second EGA or VGA monitor is needed to output error and diagnostic messages. A Hercules monitor/adapter displays the incoming seismic data.

The XDETECT program performs the following:
1) Parsing the control file (lx)
2) Data buffering (dt,dm)
3) Event triggering (t)
4) Data recording (f,o)
5) Real-time graphical display (st)
6) Processing of data using the DSP board (dsp)
7) Earthquake location (l)

Although 29 modules containing 210 functions make up the program, any of the public functions in these modules can be associated with one of the above by noting the first one to three letters of the function. The are indicated above in parens. For example the function dm_get_next_buffer() is a buffering function. Functions which do not follow the above rule fall into two classes:

a) Public initialization and utility
b) Private functions to a particular module

The modules are all compiled using the Microsoft C5.0 Compiler where the leading "m" of each of the 29 module names denotes the medium model. The "make" file XDETECT controls compilation and supports Hercules, EGA and VGA by selection of the proper graphics library using the "@" link control in the make file. Once the program is compiled, only two other files are needed for execution: the control and station files. The structure of these files is documented in the XDETECT User's Guide (Tottingham, 1990).

The general program flow is as follows: Program execution starts by parsing the control file and looking for the DT2824 driver. Important program parameters such as trigger settings are initialized to the values given in the control file, or if no value is given, to a default value. If the driver for the A-D has not been loaded the program exits. Otherwise the multiplexer (mux) ID code, which identifies the mux type, is read.

At this point several calls to ATLAB (Data Translation library for controlling the A-D) sets up the A-D for continuous DMA (Direct Memory Access) at the prescribed sample rate to extended memory buffers. ATLAB provides several callable routines to control the buffer queue in extended memory such one for copying a buffer to DOS conventional memory.

In order to simplify data indexing, the data is partially demultiplexed after it is brought from extended to conventional memory. This means that power-of-two chunks of individual channels are stored contiguously. Thus 256 samples of channel 1 follows 256 samples of channel 0.

These buffers are then stored in a linked list of structures which contains other crucial information concerning the buffered data, such as the buffer start time. This time is NOT read from the DOS system clock, but rather from the A-D clock as described below. At this point the buffers are ready for input to the plotting, triggering and digital signal processing (DSP) modules. The plotting routine provides the real time data display. The trigger routine uses a modified (see below, Event Triggering) short term - long term average (STA/LTA) to look for changes in channel amplitude while the DSP module is used for special purpose spectral monitoring. Another module does real-time location but will not be discussed in this report.

Appendix A is a program flow chart which details the calling relationship.

**DATA INPUT**
XDETECT inputs the user's network data via the control file. This file is written in a unique "input language" which allows much greater flexibility (from the user's point of view) than would be possible using simple 'C' data read routines.

As in any compiler, the input file statements are input to a lexer which constructs tokens from the input stream (control file). A parser checks the tokens for validity and the lines for correct syntax. The are about 50 tokens which the input language recognizes. These are held in the array LX_KEYWORD that must be added to if new parameter needs to be input to the program.

If an unknown symbol or token or syntax error is encountered during the compilation of a control file line, the input file compiler writes the number of the offending line to the screen. Since the control file is just inputting values to the program, the line order is not important.

**DATA BUFFERING**
The DT2824 A-D board is controlled by the Data Translation ATLAB library which allows for a variety of ways to input data to the a program via 'C' callable routines. For example a single data point from a single channel may be input or, as in the case of XDETECT, a series of channels may be input. For real-time programing it is desirable that data be read without CPU intervention. This leaves the CPU free to process the data, control the recording of events, update the display etc.

This is made possible before XDETECT runs when the user sets up the config.sys file for extended memory buffering using the /E option followed by the number of kilobytes. A typical statement might be:

device = c:\atlab\atldrv.sys /E 2048.

In this case the driver will have 2 Mbytes of extended memory available for buffers. The amount needed depends on three factors:

a) the number of channels (1 - 128)
b) pre-event memory, seconds
c) sampling rate, samples per second
d) disk speed (access time, msec)

XDETECT must be able to write out all the pre-event data to disk before the DMA process overwrites the buffer which caused the event. For example, a PC/AT digitizing 128 channels with a slow (>30 msec) hard disk and 60 seconds of pre-event memory may need several megabytes of extended memory available for the DMA buffers. On the other hand, a 33 MHz 486-based PC with a fast (16 msec) hard disk might only need 1 Mbyte.

When XDETECT starts running ATLAB allocates the extended memory buffers mentioned above until it runs out of memory. These buffers are linked in a queue or transfer list. The sampling rate is then calculated given the user's desired sampling rate. Usually, the DT2824 can only sample at a rate close to the desired sampling rate due to limitations in the hardware (Data Translation DT2821 Manual, 1987). If an exact sampling frequency is needed XDETECT and the hardware allow for an external sampling clock. If the internal clock is used it is accurate to about 0.01%.

XDETECT then sets up the board for continuous DMA to extended memory at the proper sampling rate via several calls to the ATLAB routines. Once the DMA has started the program must process buffers fast enough so a buffer is never re-filled before it has been examined by XDETECT. If this occurs, a buffer overflow error is generated and the program quits.

This error can happen for a variety of reasons:
a) The CPU cannot keep up. For example if 128 channels were digitized at 200 Hertz on a 8 Mhz PC/AT the CPU could not calculate the STA/LTA fast enough.
b) The DSP module places too high a demand on the CPU. Even though the DSP coprocessor runs at 25 MFLOPS, some of the information provided is processed by the CPU. If more than 32 channels are handled by a 8 MHZ PC/AT, a buffer overflow might occur.
c) The disk is too slow (see above).

XDETECT runs asynchronously with the ATLAB DMA operation. Thus XDETECT may fall behind temporarily, but it must be able to catch up in the end. With a large extended memory queue, XDETECT might take several minutes to finally crash. The solutions to buffer overflows are to reduce the sampling rate or number of channels, or increase the CPU and disk speeds. A larger extended memory queue may only delay the inevitable.

The data are written to extended memory and retrieved using an ATLAB function which copies extended memory buffers to conventional DOS memory. These buffers are in multiplexed format. XDETECT uses the function to_demux() to partially demultiplex the data. Once demuxed, the data is again kept in extended memory (but in a different area from that used by ATLAB) using the PowerStor memory manager. The amount of data kept in extended memory depends on the amount of pre-event memory specified in the control file. An approximation of the amount needed is given by:

memory required = number_channels*sampling_rate*pre-event_memory*2

The data at this point is no longer raw buffers of data but structured "Q_BUFFER" data in a linked list. Each buffer contains a pointer to the next buffer as well as a buffer time. This time is not kept by the DOS clock, by rather by the program itself. XDETECT reads the DOS clock upon startup and after that point counts buffers. Since the number of samples in a buffer is known as well as the sampling rate, the current time is merely the start time plus the number of buffers times the time per buffer. This time is written in the Q_BUFFER structure.

The routines dm_get_head_buffer(), dm_get_buffer_time() and dm_get_next_buffer() are some of the public functions which can be used by other modules which need to access buffered data.

A problem inherent in the hardware occurs when the channel counter on the A-D or the bank[1] counter on the multiplexer loses synchronization with the software. Although a rare occurrence, a software modification of the DT2824 driver allows such "bank skips" to be detected (channel skips cannot be detected). XDETECT uses the digital input feature of the DT2824 to read the multiplexer ID and bank number. (If the mux ID is not an allowed value, an error message is given and the program exits. The ID is checked only once at startup). If the bank number read is not correct, the multiplexer is reset to regain synchronization with the A-D.

This detection works on the fact that at one point in each digitizing pass the mux counter and channel counter must both be zero. Since the A-D digitizes 16 channels per pass and the multiplexer feeds the A-D banks of 16 channels, when the A-D goes back to the first channel, the mux must also be on the first bank. If this is not the case, a bank skip has occurred and the mux is reset. The Q_BUFFER structure has a member of type FLAG called "bank_switched" which flags the buffers as defective.

The causes of bank and channel skips are static or ac power glitches. These problems can be minimized by putting the PC system on a UPS by itself and by following good grounding procedures.

**EVENT TRIGGERING**

XDETECT uses a modified STA/LTA triggering algorithm based on first difference calculations. Three threshold are provided to control triggering and recording as described in the XDETECT User's Guide (Tottingham, 1990). For each channel a difference between two consecutive data points is calculated. The absolute value of this difference is the new value in a short-term-average (STA) and long-term-average (LTA) calculation. The averaging time for the LTA is usually chosen to be an order of magnitude longer than that for the STA so that when an abrupt change in signal energy occurs, the STA

[1]The A-D digitizes up to 16 channels. The multiplexer fools the A-D by switching in "banks" of 16 channels for the A-D to digitize. The software treats each group of 16 channels as a bank.

can change rapidly in relation to the LTA. Slow signal buildups will be followed by the LTA, leaving the STA/LTA ratio unchanged.

Since many noise sources affect only single channels, a multi-channel time window criterion for recording is employed. This means that for an event to be qualified for possible recording it needs to generate a preset number of triggers within a time window. Also it is required that a recorded event must possess a minimum energy duration on the active channels. This energy criterion tends to eliminate short telemetry dropouts and noise spikes as well as very small events. Events which satisfy the above two criteria are then recorded.

The STA and LTA are actually running averages which approximate the real average but are easier to update:

```
sta += (abs(difference_value) - sta)>>K1
lta += (abs(difference_value) - lta)>>K2
```

Here ">>" is the integer division-by-a-power-of-two operator implemented using bit shifting. Ignoring the non-linearity of the absolute value operator, these equations can be rewritten as follows:

```
y(n) = (1/K) * x(n) +  ((K-1)/K) * y(n-1)
```

where x represents the input (difference or data point) value, y the output STA or LTA and K is either K1 or K2. The index n is the present time epoch, n-1 the previous time epoch, etc. This difference equation implements a normalized low-pass filter whose cut-off frequency is controlled by K. A larger K means a lower the cut-off frequency and a longer averaging time. In order to determine K if the cutoff frequency is specified (or visa-versa), the following equation must be solved for the unknown variable:

```
a**2 + (2 * cos(w*T) - 4)*a + 1 = 0        (Cadzow, 1973)
```

where a = (K-1)/K, w = cut-off frequency in radians and T = sampling period in seconds.

The STA has a higher cut-off frequency than the LTA, resulting in a faster response time. In practice the K value for the LTA usually is 100 times greater than that for the STA.

Since XDETECT STA/LTA triggering approximates the above calculations, erratic operation can result for low input signals due to the fixed-point arithmetic employed. For example, if the input were 15 counts (after removing the offset), the STA were zero and K = 16, then the STA would remain at zero indefinitely. This problem was solved in MDETECT by updating the remainder of each division as shown below:

```
rsta += ( abs(difference_value - sta)) % K
```

Then the STA is then updated according to the formulas:

sta += rsta >> K1  and  rsta %= K1

Here "%" is the modulus operator. The calculations for the LTA are the same as for the STA with the exception that K2 is used in place of K1. Additionally, LTA is also never allowed to fall below a minimum value to prevent dividing by zero when the STA/LTA ratio is computed.

There are two additional problems in the XDETECT algorithm, although it works reasonably well for dense networks and local earthquakes.
a) The first difference calculation includes implicit filtering. XDETECT will generally not trigger on signals less than 1 or 2 Hz.
b) If the first difference is disabled (V2.03) offsets produced by the telemetry can desensitize the trigger since they are not removed by XDETECT.

These problems will be cleared up in version 3.0, therefore no further discussion of version 2.01 - 2.03 triggering will be included in this report.

**DATA RECORDING**
XDETECT records five types of files:
a) triggered events - Earthquake or noise events recorded due to the event passing all the trigger requirements
b) calibrations - A1VCO (see below) station calibrations, usually one per station per day.
c) free-run events - Initiated and stopped by the operator toggling control-F from the keyboard.
d) log of events - A summary of all triggered activity
e) continuous FFT files - Averaged spectra for a the first NCHAN channels, where NCHAN is a number less than the number of channels being digitized (see below).

The first four file types are controlled by the mfile.c and mlog.c modules. The continuous FFT files are written from the mdsp.c module and is discussed there. The mfile and mlog modules contain routines which create the proper path names used to write the files to disk. This includes routines to increment the event file and find the calibration channel number. Event files are named using base 36 arithmetic (a .. z.. 0..9). for the last two characters of the DOS eight character name (YYMMDDNN). Here YY is the year, MM the month, DD the day and NN, a two-digit base-36 number. This arithmetic is done in the mbase36.c module. Since calibrations are transmitted only every 24 hours there is no issue of uniqueness so these two digits are used for the calibration channel.

The function f_write_buffers controls the data recording via a 'C' switch statement. The different event types above need to record various numbers of buffers, depending on the file type. For example the free-run event duration is controlled by the keyboard, the calibration duration by the XDETECT clock and triggered events by the event itself. The switch statement takes an argument which determines ultimately many buffers are recorded. The value of the argument passed controls whether a file needs to be opened, appended to or closed.

**REAL-TIME DISPLAY**
One of the major improvements of XDETECT over many other seismic event recording systems is the real-time display. This display allows seismologists and technicians to view the quality of the incoming data and evaluate the program's operation (or lack thereof!).

Graphics is handled with calls to the Symmetric Research graphics library. This library is very fast compared to the 'C' graphics functions and supports VGA, EGA and the Hercules standards. XDETECT starts by calling the function border() which draws a box on the screen outlining the display size available. Next, header() draws the status bar. The arguments that header() takes are used to write the program information such as the version number. Calls to the function plot() and plotmod() plot the waveforms on the screen. These waveforms are 512-sample windows of the last buffer, with either 16 or 32 channels displayed.

**DIGITAL SIGNAL PROCESSOR SUPPORT**
Although a 50 MHz 80486-based PC is about 100 times faster than the 4.77 Mhz PC, this processing power is still two orders of magnitude less than that offered by certain boards which employ one of the popular DSP chips. XDETECT supports the Symmetric Research DSP32C board based on an ATT floating point chip which executes 25 million floating point instruction per second with 24-bit resolution. This allows a 512-point FFT to be computed in about 10 msec. Symmetric Research provides a math library callable from 'C' with over 20 functions. Specialized operations can be programmed in the ATT assembly language for maximum speed and flexibility.

The program module MDSP.C controls the DSP co-processor board. To access the incoming data, the XDETECT function dm_get_head_buffer() needs to be called. This function returns a far pointer to a structure of type Q_BUFFER. This structure is defined in the header file mqueue.h. Once the pointer is returned, the data in the structure is copied to a scratch array for processing to perform two useful functions:
1) calibration detection
2) continuous spectral band recording

Although most seismic stations have a calibration cycle, it is not possible to detect these calibrations using conventional STA/LTA triggering since they occur asynchronously on single channels. For the Alaska A1VCO (Rogers et. al, 1980) the stations have a 21.25 Hz preamble which is detected using spectral analysis. A call to dm_get_head_buffer() and dm_get_next_buffer() is used to construct a 512-point array. A call to spectrum() in the math library produces an absolute amplitude frequency domain representation of the buffer. Only the lower 256 lines are unique so the upper 256 lines are ignored. Each line represents a frequency according to the relation:

frequency = line_number* sample_rate/NPTFFT

where NPTFFT is usually 512 and the sample frequency is obtained from a call to dt_get_digitization_rate(). For a sampling rate of 100 Hz, the 108,109 and 110 lines surround the 21.25 Hz calibration frequency. When the average of the sum of these three lines exceeds the average of all the lines by the factor CALTHR, a calibration is detected and recording is turned on. This technique is very effective in capturing calibrations with few false triggers and few misses.

Once the data is converted to frequency lines it is possible to record a representation of the incoming data continuously. This representation is derived by averaging the FFT data into a small number of bands and then averaging over time. MDSP.C uses 16 bands which are averaged about every minute (24, 2.56 second buffers) to a single number per band. These numbers are written to a .FFT file, where the prefix of each file is in the YYMMDDHH format. Each file contains about one hour's worth of FFT data. This data has proven useful in monitoring volcanoes.

The user selects the bands to be monitored in the control file, specifying the band limits in Hertz. If the band limits do not fall exactly on a spectral line number, then the line numbers are calculated such that the band limit (in Hertz) corresponds to the line just above the limit. If non-overlapping bands are specified, then no line will be included in two bands. Given a 512-point buffer and a sampling rate of 100 Hz, the frequency resolution is about 0.2 Hz. This resolution can be increased by increasing the size of the buffer. This will, however, reduce the time resolution.

The above is represented by the following equation:

line_limit[i] = (int)(freq_limit[i]*NPTFFT)/digitization_rate+1,

where:
line_limit is the frequency expressed as a line number,
freq_limit is frequency in Hz as given by the user,
NPTFFT is the number of points in the FFT (usually 512),
(int) converts float to integer,
and digitization_rate is the sampling rate (usually about 100).
This equation is calculated for the upper and lower limit for each

band.

The band data is also converted to ascii and written out the parallel port every minute for real-time display (Murray, 1989).

**XDETECT SUPPORT**

The USGS is interested in suggestions for improvements and possible bugs in XDETECT. Please write to:

U. S. Geological Survey
c/o John Rogers
345 Middlefield Rd.
Menlo Park, CA 94025
USA

## REFERENCES


Tottingham, D.M., Lee, W.H.K., and Rogers, J.A. 1989. User Manual for MDETECT. In: Lee, W.H.K., editor, Toolbox for seismic data acquisition, processing and analysis. Published by International Association of Seismology and Physics of the Earth's Interior.

Tottingham, D.M., and Lee, W.H.K. 1989. User Manual for XDETECT. In: Lee, W.H.K., editor, Toolbox for seismic data acquisition, processing and analysis. Published by International Association of Seismology and Physics of the Earth's Interior.

Ellis, J.O. 1989. Expanding the input multiplexer for the Data Translation, Inc. Model DT2821 analog-to-digital converter, U.S. Geol. Surv. Open File Rep.,89-201.

Rogers, J. A. 1989. Frequency-domain detection of seismic signals using a DSP coprocessor board. In: Lee, W.H.K., editor, Toolbox for seismic data acquisition, processing and analysis. Published by International Association of Seismology and Physics of the Earth's Interior.

Tottingham, D.M. 1990. XDETECT user's guide version 1.93, Computer Science Dept., Calif. Polytechnic State Univ. (unpublished)

Data Translation 1987. User Manual for DT2821 Series, Marlboro, MA

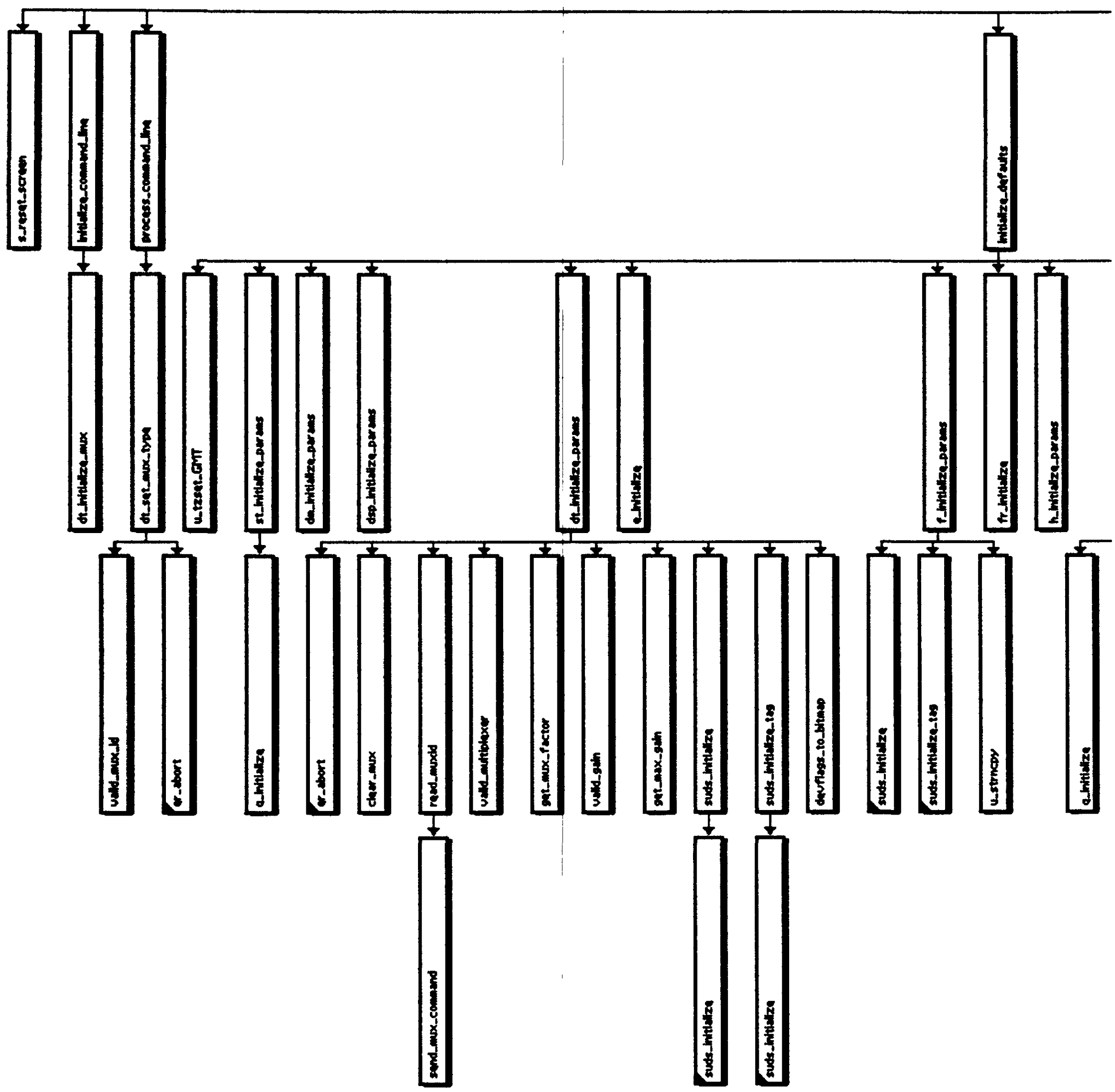
Cadzow, J.A. 1973. Discrete Time Systems, Prentice-Hall, Englewood Cliffs, N.J.

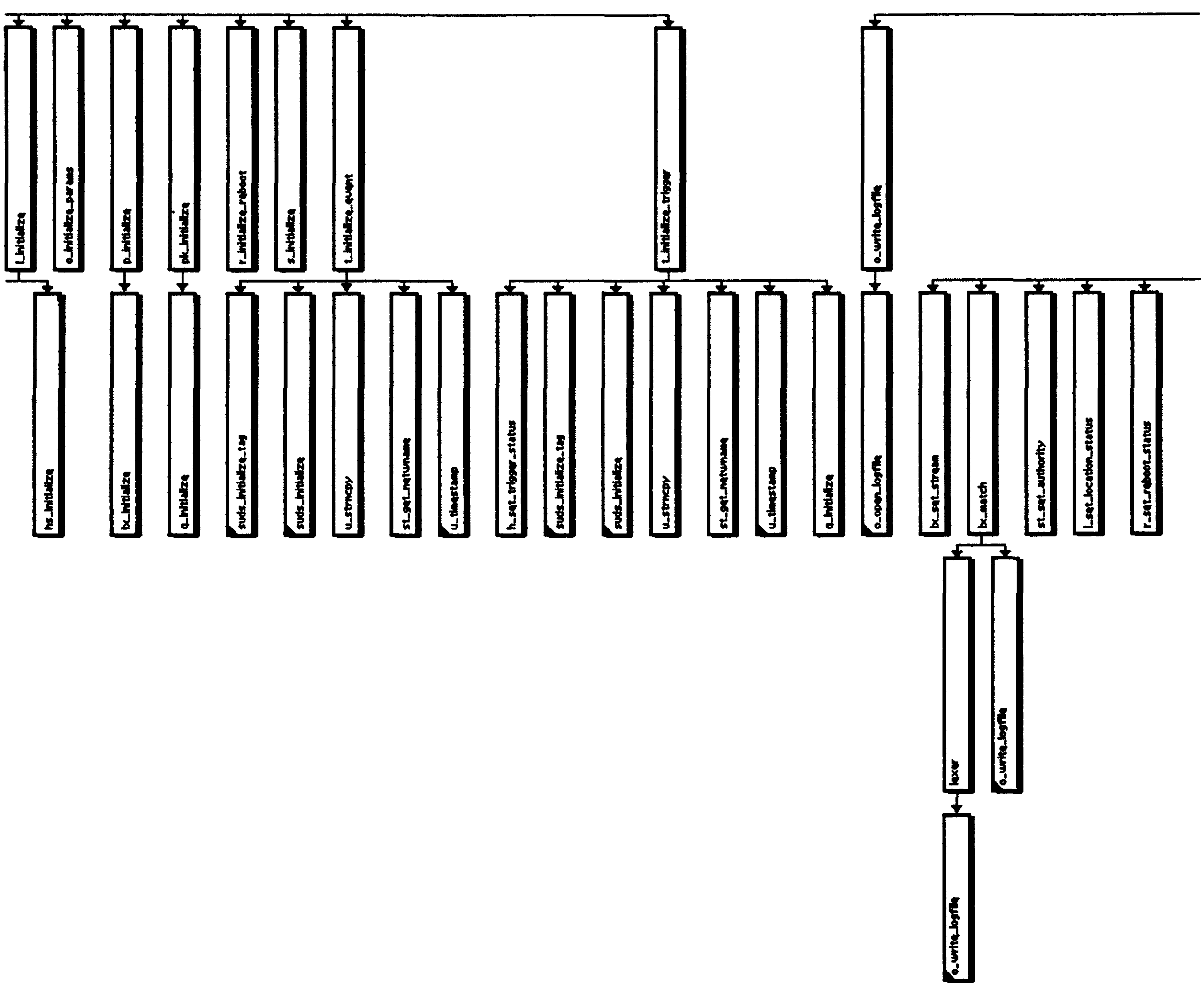
Rogers, J.A., Maslak, S., Lahr, J.C. 1980. A seismic electronic system with automatic calibration and crystal reference. U.S. Geol. Surv. Open File Rep. 80-324.

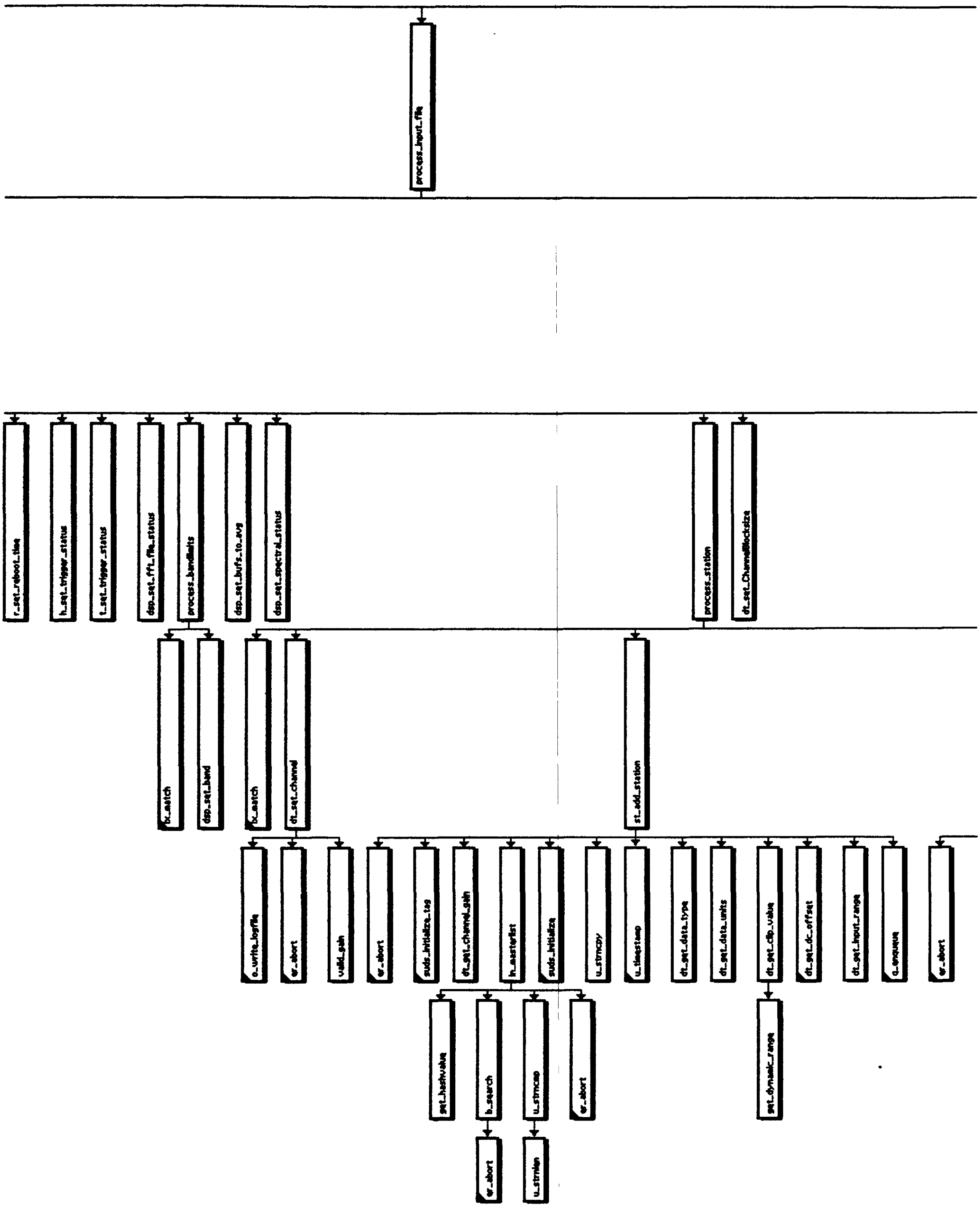
Murray, T. L. 1989. A user's guide to the PC based time-series data management and plotting program BOB, U.S. Geol. Surv., Open File Rep. 90-634-A.

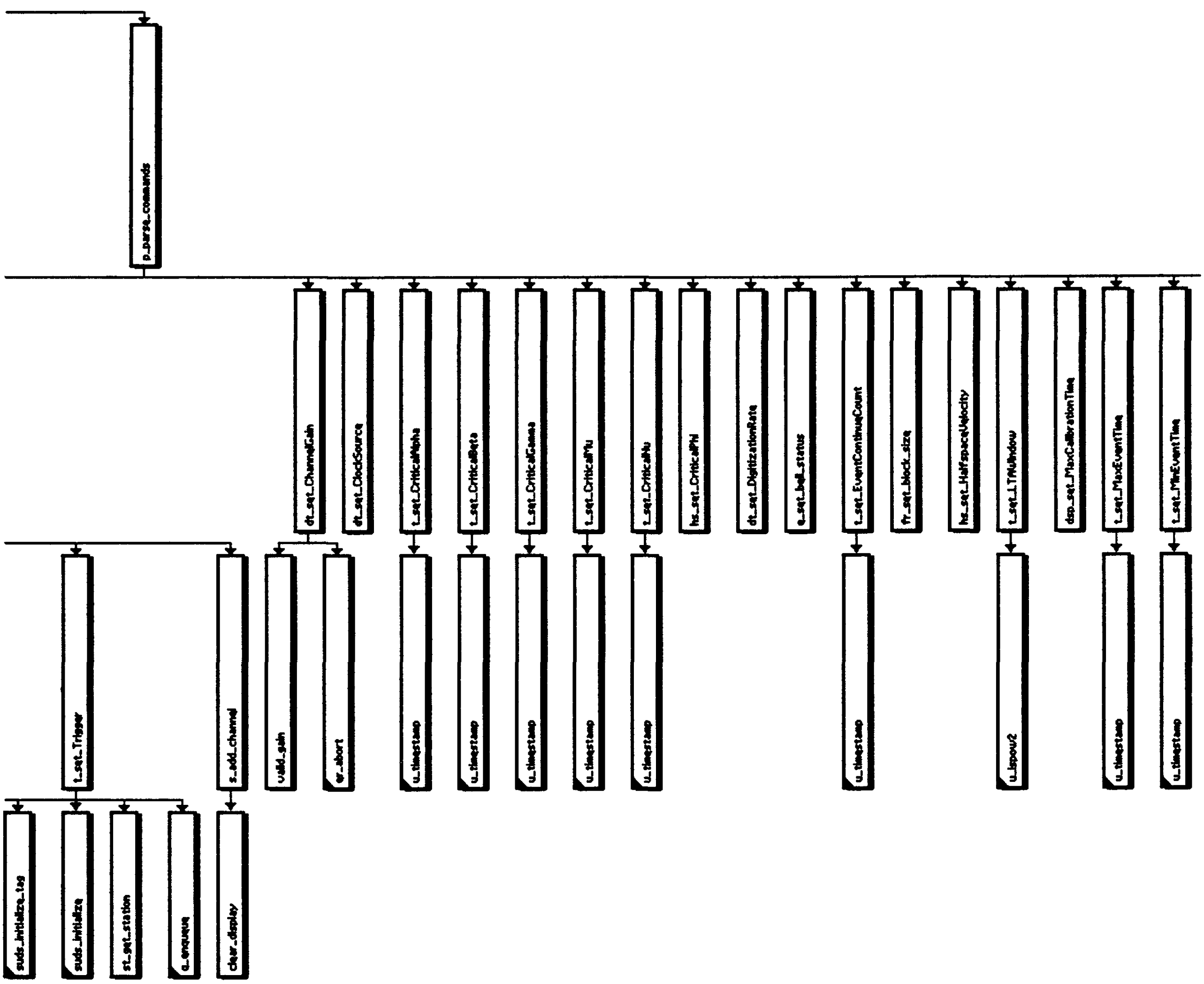
# XDETECT 2.01

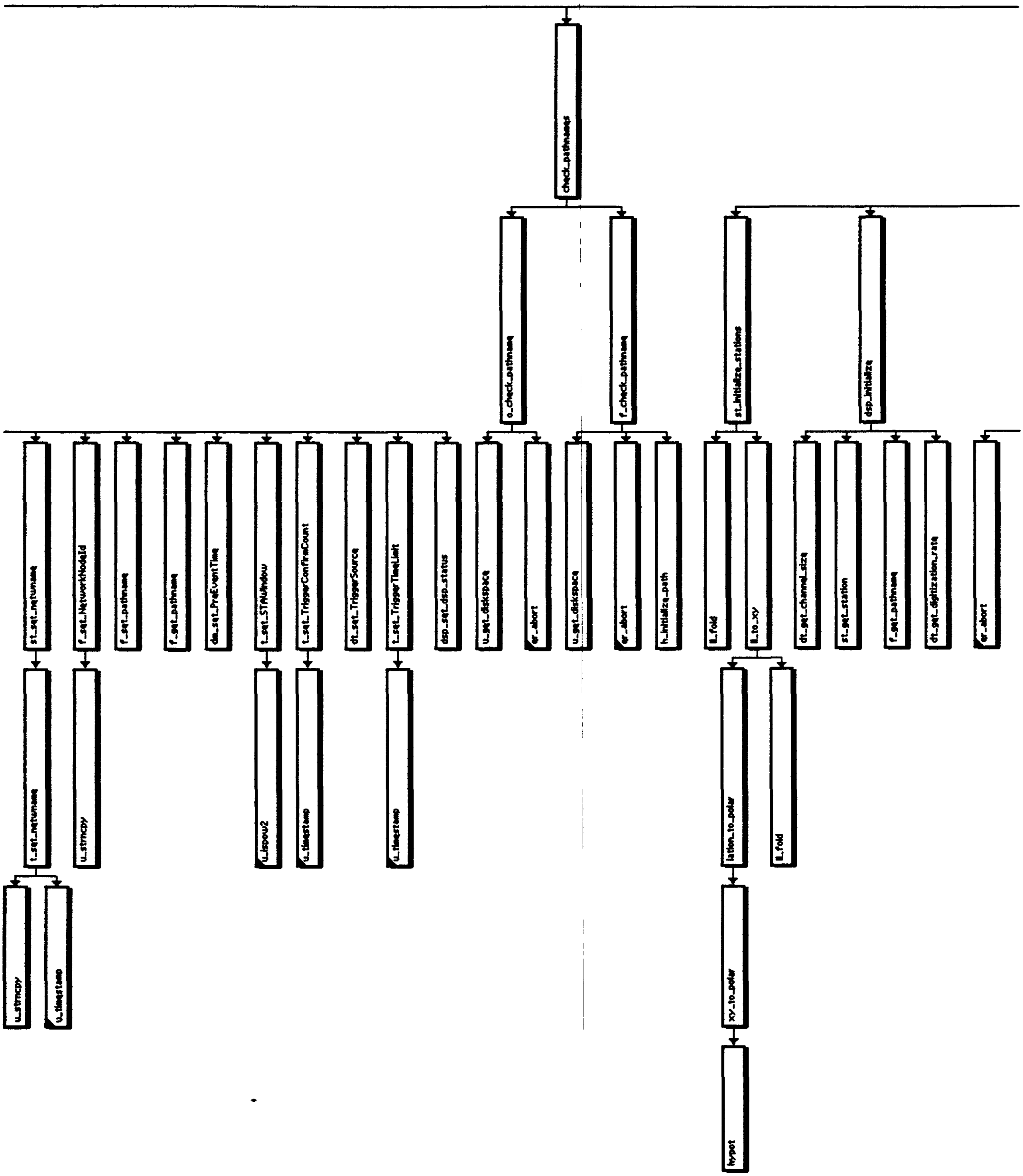
APPENDIX

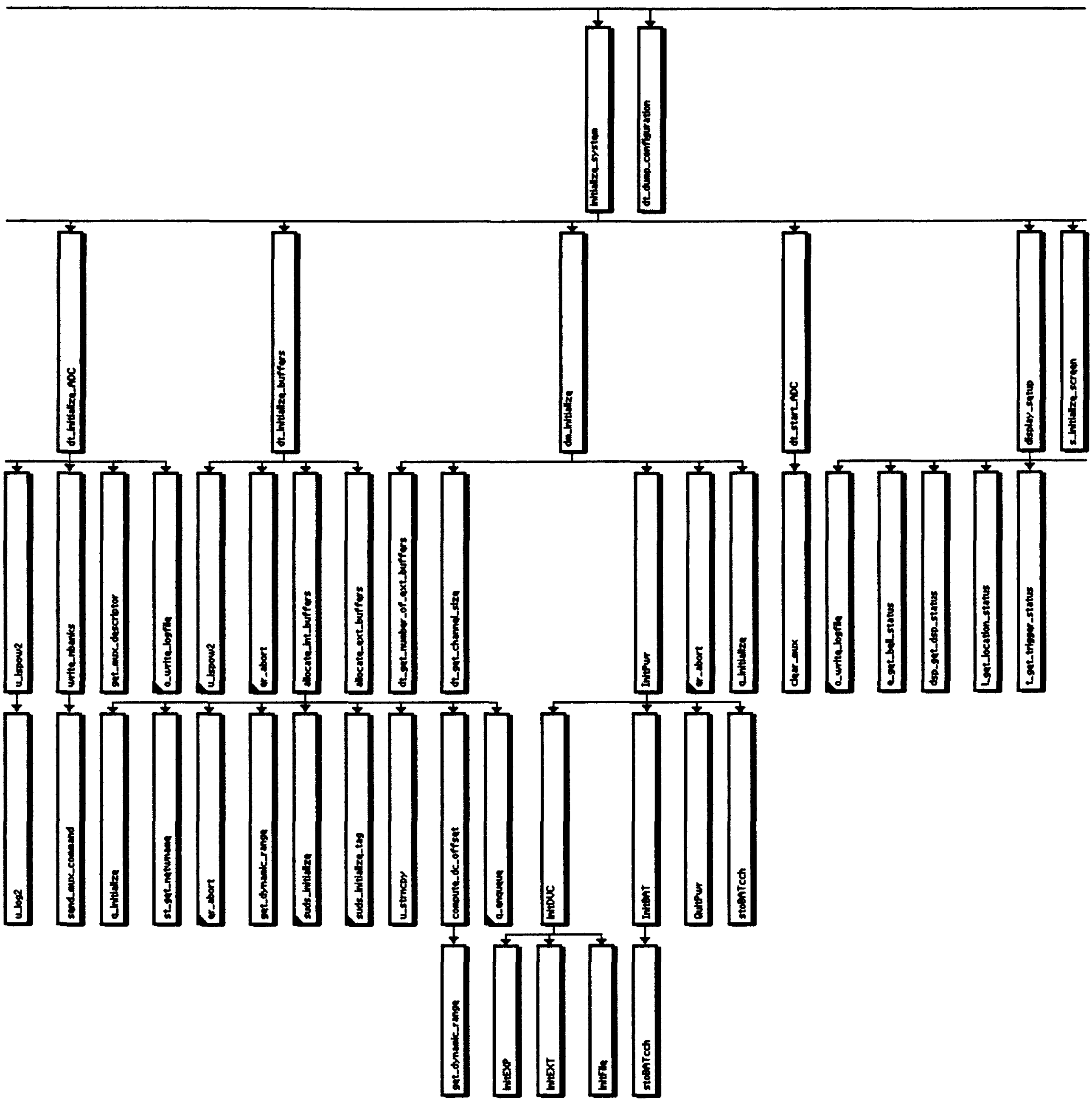


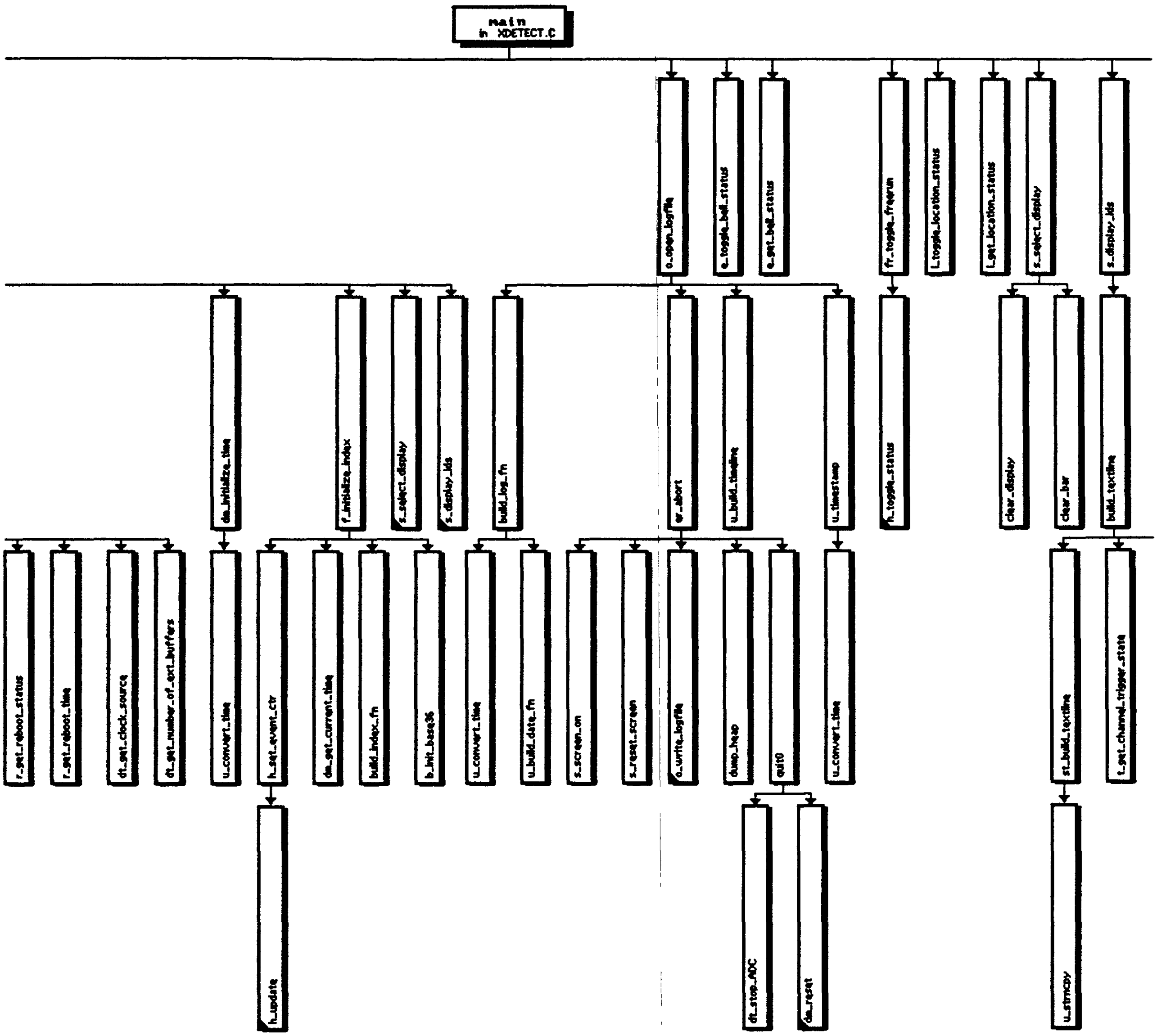

l_initialize
o_initialize_params
p_initialize
pk_initialize
r_initialize_reboot
s_initialize
t_initialize_event
t_initialize_trigger
o_write_logfile
hs_initialize
bc_initialize
q_initialize
suds_initialize_tag
suds_initialize
u_strncpy
u_timestamp
h_set_trigger_status
suds_initialize_tag
suds_initialize
u_strncpy
u_timestamp
q_initialize
o_open_logfile
bc_set_stream
bc_match
st_set_authority
l_set_location_status
r_set_reboot_status
lexer
o_write_logfile
o_write_logfile

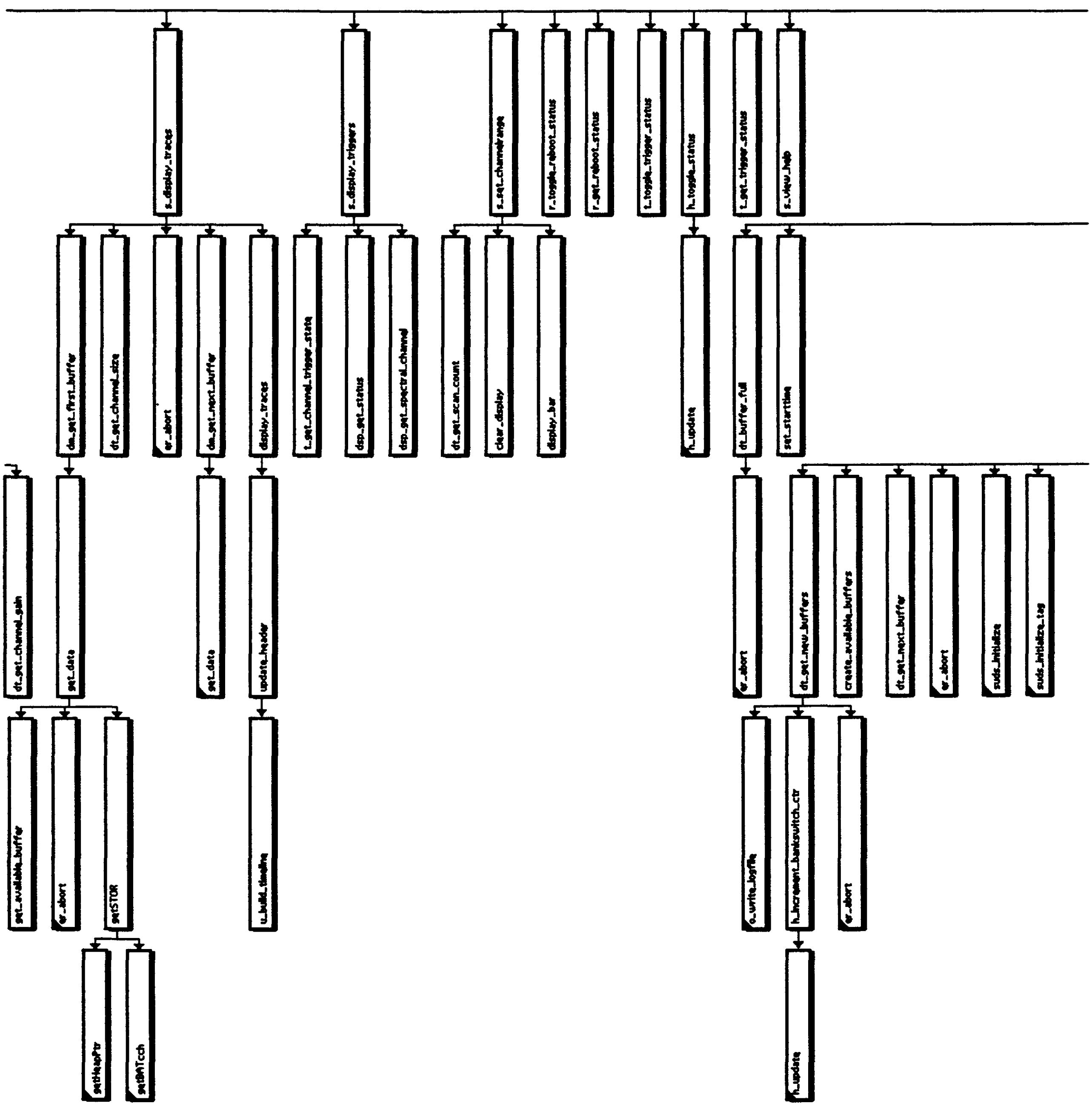

process_input_file
r_set_reboot_time
h_set_trigger_status
t_set_trigger_status
dsp_set_fft_file_status
process_bandlimits
dsp_set_bufs_to_avg
dsp_set_spectral_status
process_station
dt_set_ChannelBlocksize
bt_match
dsp_set_band
bt_match
dt_set_channel
st_add_station
o_write_logfile
er_abort
valid_gain
er_abort
suds_initialize_tag
dt_set_channel_gain
h_masterlist
suds_initialize
u_strncpy
u_timestamp
dt_set_data_type
dt_set_data_units
dt_set_clip_value
dt_set_dc_offset
dt_set_input_range
q_enqueue
er_abort
set_hashvalue
b_search
u_strncmp
er_abort
get_dynamic_range
er_abort
u_strnlen

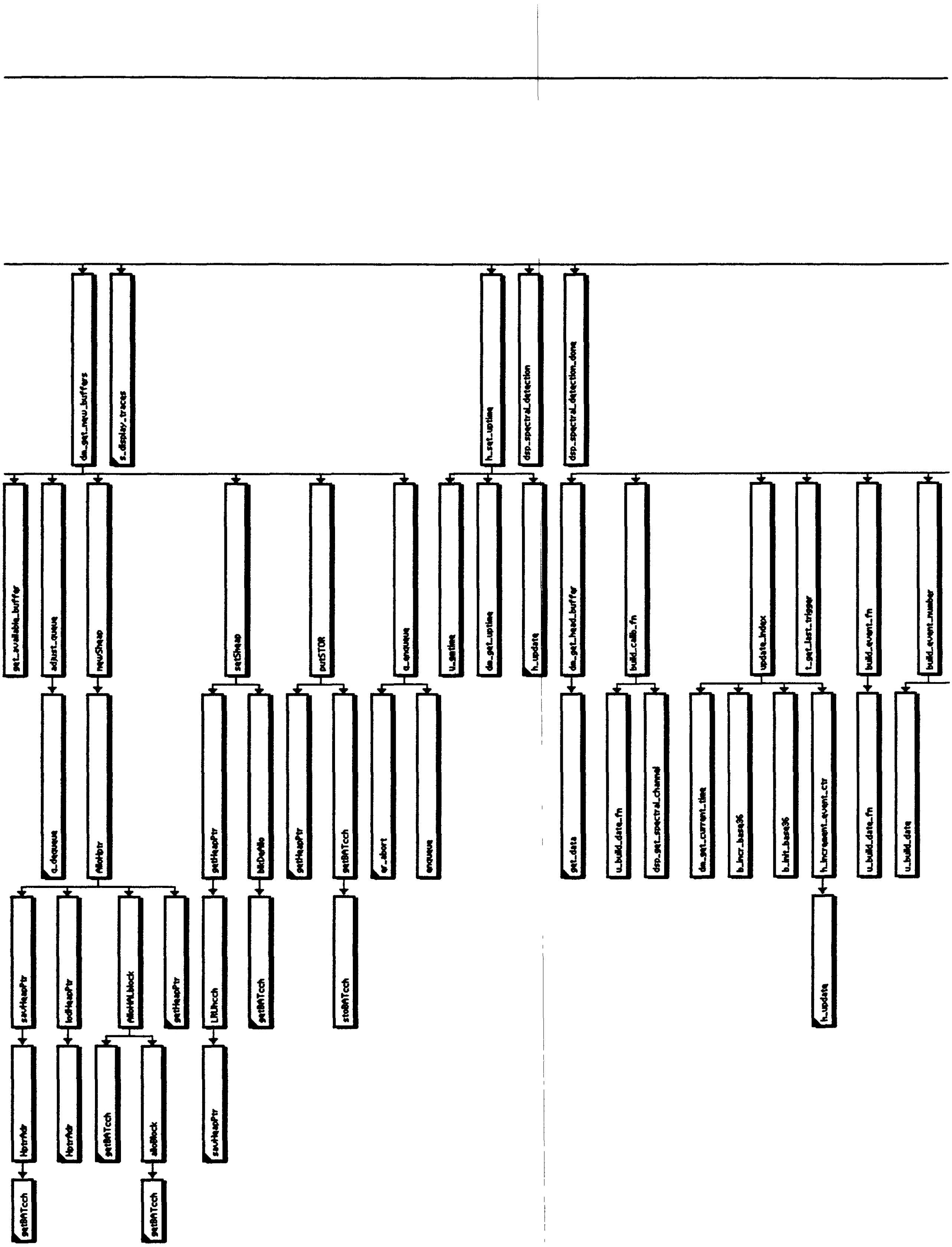

p_parse_commands
dt_set_ChannelGain
dt_set_ClockSource
t_set_CriticalAlpha
t_set_CriticalBeta
t_set_CriticalGamma
t_set_CriticalMu
t_set_CriticalNu
hs_set_CriticalPN
dt_set_DigitizationRate
e_set_bell_status
t_set_EventContinueCount
fr_set_block_size
hs_set_HalfspaceVelocity
t_set_LTAWindow
dsp_set_MaxCalibrationTime
t_set_MaxEventTime
t_set_MinEventTime
t_set_Trigger
s_add_channel
valid_gain
er_abort
u_timestamp
u_timestamp
u_timestamp
u_timestamp
u_timestamp
u_timestamp
u_ispow2
u_timestamp
u_timestamp
suds_initialize_tag
suds_initialize
st_get_station
q_enqueue
clear_display

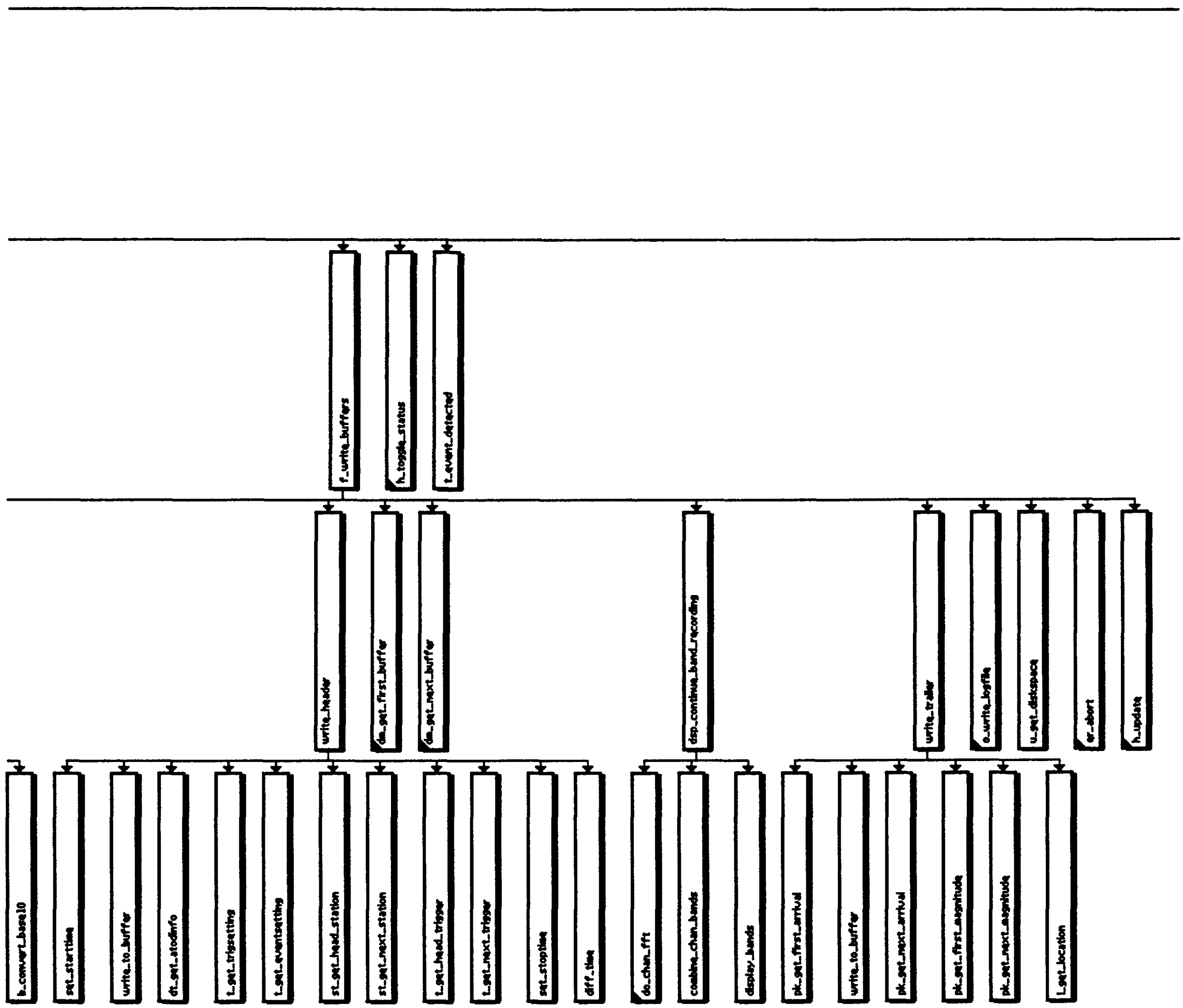

check_pathnames
o_check_pathname
f_check_pathname
st_initialize_stations
dsp_initialize
st_set_netunname
f_set_NetworkNodeId
f_set_pathname
f_set_pathname
da_set_PreEventTime
t_set_STAWindow
t_set_TriggerConfirmCount
dt_set_TriggerSource
t_set_TriggerTimeLimit
dsp_set_dsp_status
u_get_diskspace
er_abort
u_get_diskspace
er_abort
h_initialize_path
ll_fold
ll_to_xy
dt_get_channel_size
st_get_station
f_get_pathname
dt_get_digitization_rate
er_abort
t_set_netunname
u_strncpy
u_ispow2
u_timestamp
u_timestamp
lation_to_polar
ll_fold
u_strncpy
u_timestamp
xy_to_polar
hypot

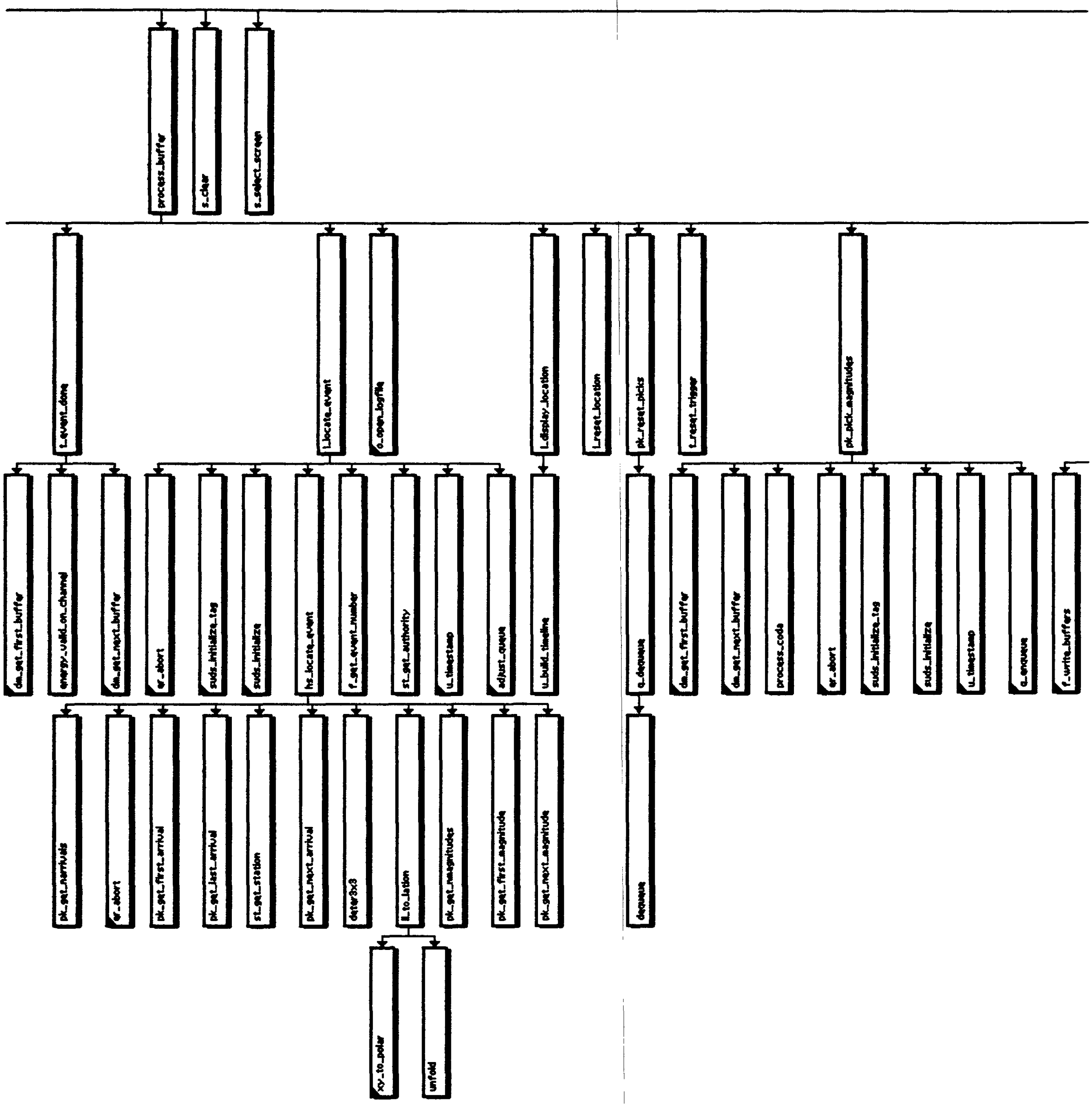

initialize_system
dt_dump_configuration
dt_initialize_ADC
dt_initialize_buffers
da_initialize
dt_start_ADC
display_setup
s_initialize_screen
u_ispow2
write_nbanks
set_aux_descriptor
o_write_logfile
u_ispow2
er_abort
allocate_int_buffers
allocate_ext_buffers
dt_get_number_of_ext_buffers
dt_get_channel_size
InitPwr
er_abort
q_initialize
clear_aux
o_write_logfile
e_get_bell_status
dsp_get_dsp_status
l_get_location_status
t_get_trigger_status
u_log2
send_aux_command
q_initialize
er_abort
get_dynamic_range
suds_initialize
suds_initialize_tag
u_strncpy
compute_dc_offset
q_enqueue
InitDUC
InitBAT
QuitPwr
stoBATcch
get_dynamic_range
InitEXP
InitEXT
InitFile
stoBATcch

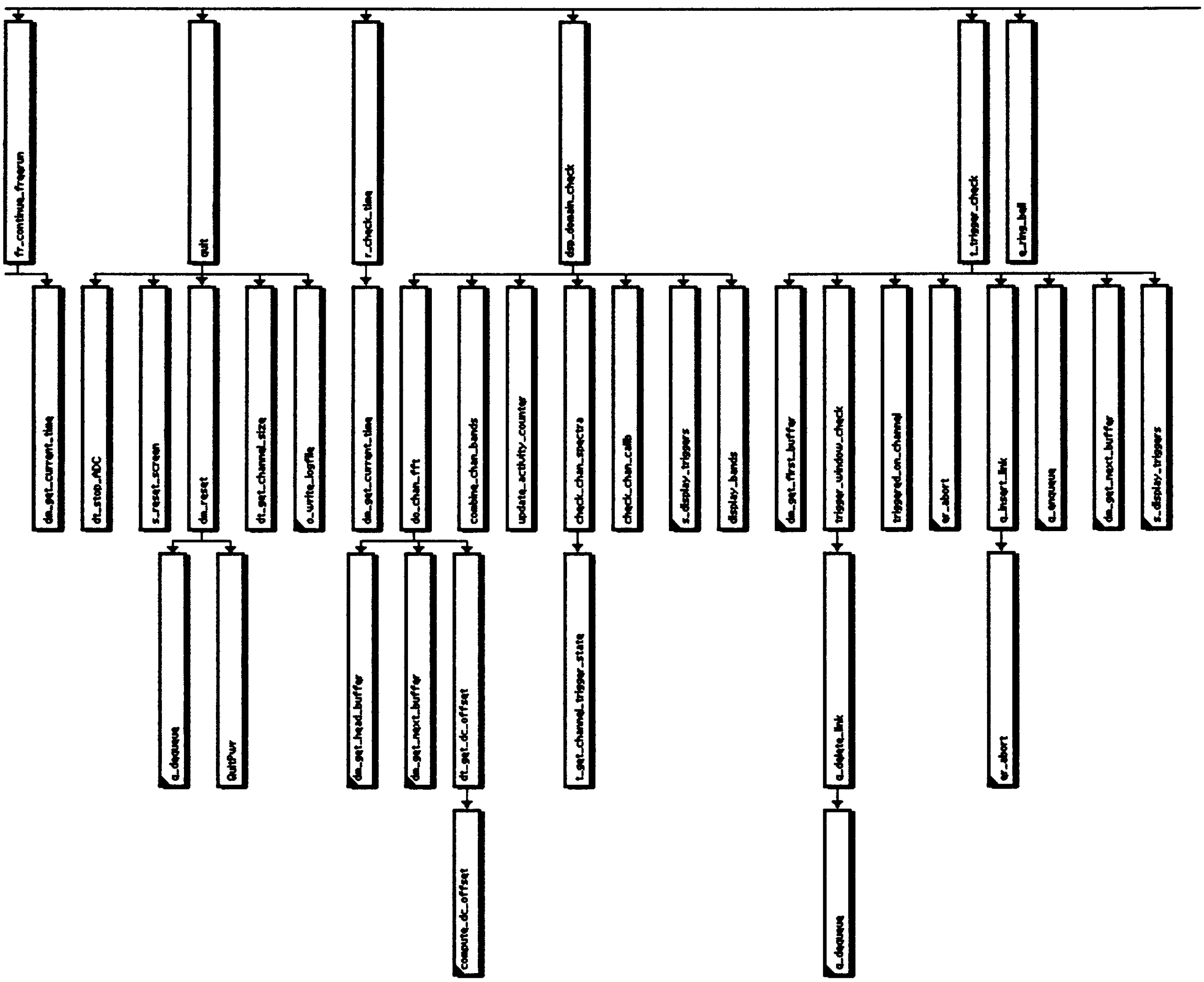

main
in XDETECT.C
o_open_logfile
a_toggle_bell_status
a_get_bell_status
fr_toggle_freerun
l_toggle_location_status
l_get_location_status
s_select_display
s_display_ids
da_initialize_time
f_initialize_index
s_select_display
s_display_ids
build_log_fn
er_abort
u_build_timeline
u_timestamp
h_toggle_status
clear_display
clear_bar
build_textline
r_get_reboot_status
r_get_reboot_time
dt_get_clock_source
dt_get_number_of_ext_buffers
u_convert_time
h_set_event_ctr
da_get_current_time
build_index_fn
b_init_base36
u_convert_time
u_build_date_fn
s_screen_on
s_reset_screen
o_write_logfile
dump_heap
quit0
u_convert_time
st_build_textline
t_get_channel_trigger_state
h_update
dt_stop_ADC
da_reset
u_strncpy

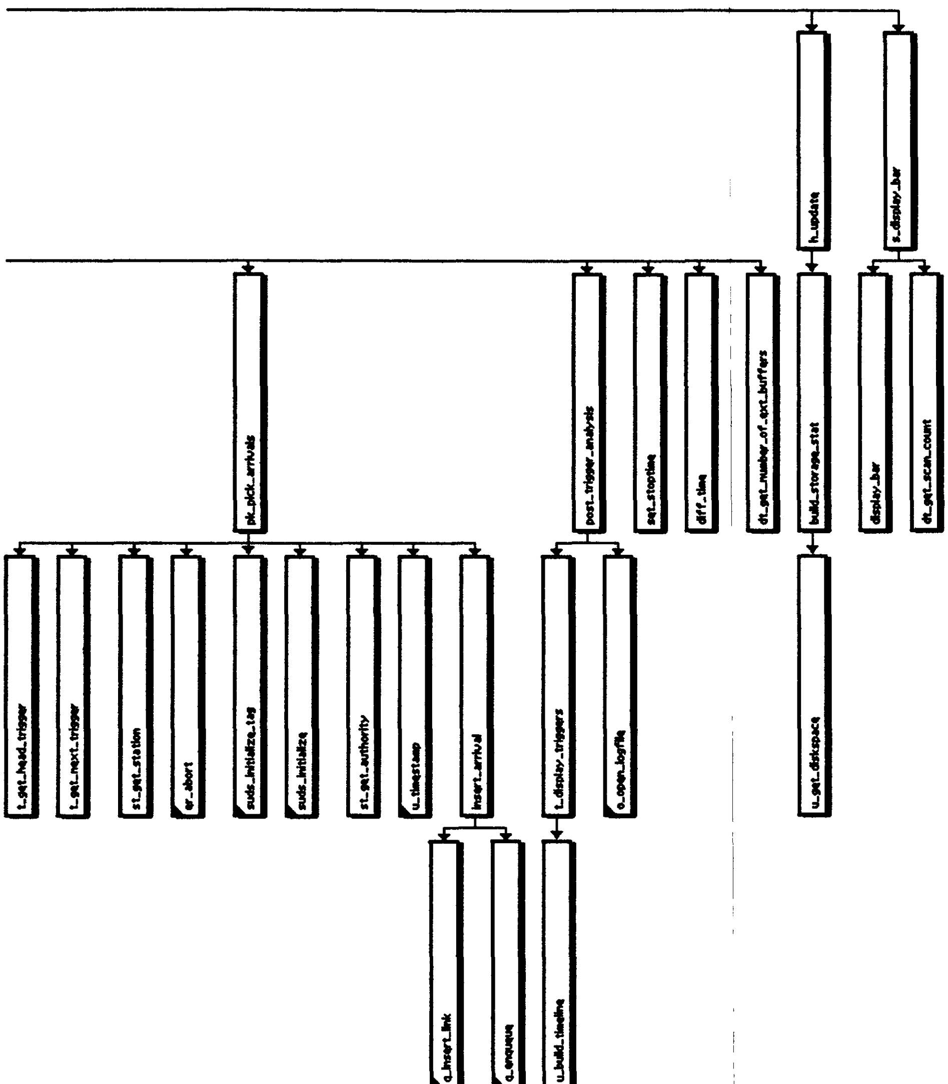

s_display_traces
s_display_triggers
s_set_channelrange
r_toggle_reboot_status
r_get_reboot_status
t_toggle_trigger_status
h_toggle_status
t_get_trigger_status
s_view_help
da_get_first_buffer
dt_get_channel_size
er_abort
da_get_next_buffer
display_traces
t_get_channel_trigger_state
dsp_get_status
dsp_get_spectral_channel
dt_get_scan_count
clear_display
display_bar
h_update
dt_buffer_full
set_starttime
dt_get_channel_gain
get_data
get_data
update_header
er_abort
dt_get_new_buffers
create_available_buffers
dt_get_next_buffer
er_abort
suds_initialize
suds_initialize_tag
get_available_buffer
er_abort
getSTOR
u_build_timeline
o_write_logfile
h_increment_bankswitch_ctr
er_abort
getHeapPtr
getBATCch
h_update


da_get_new_buffers
s_display_traces
h_set_uptime
dsp_spectral_detection
dsp_spectral_detection_done
get_available_buffer
adjust_queue
newSheap
setSheap
putSTOR
q_enqueue
u_gettime
da_get_uptime
h_update
da_get_head_buffer
build_calib_fn
update_index
t_get_last_trigger
build_event_fn
build_event_number
q_dequeue
getHeapPtr
setBATcch
er_abort
enqueue
get_data
u_build_date_fn
dsp_get_spectral_channel
da_get_current_time
b_incr_base36
b_init_base36
h_increment_event_ctr
u_build_date
savHeapPtr
lodHeapPtr
getHeapPtr
stoBATcch
h_update
setBATcch
savHeapPtr
setBATcch
setBATcch


f_write_buffers
h_toggle_status
t_event_detected
write_header
dm_get_first_buffer
dm_get_next_buffer
dsp_continue_band_recording
write_trailer
e_write_logfile
u_get_diskspace
er_abort
h_update
b_convert_base10
get_starttime
write_to_buffer
dt_get_atodinfo
t_get_trigsetting
t_get_eventsetting
st_get_head_station
st_get_next_station
t_get_head_trigger
t_get_next_trigger
get_stoptime
diff_time
do_chan_fft
combine_chan_bands
display_bands
pk_get_first_arrival
write_to_buffer
pk_get_next_arrival
pk_get_first_magnitude
pk_get_next_magnitude
l_get_location


process_buffer
s_clear
s_select_screen
t_event_done
l_locate_event
o_open_logfile
l_display_location
l_reset_location
pk_reset_picks
t_reset_trigger
pk_pick_magnitudes
da_get_first_buffer
energy_valid_on_channel
da_get_next_buffer
er_abort
suds_initialize_tag
suds_initialize
hs_locate_event
f_get_event_number
st_get_authority
u_timestamp
adjust_queue
u_build_timeline
q_dequeue
da_get_first_buffer
da_get_next_buffer
process_code
er_abort
suds_initialize_tag
suds_initialize
u_timestamp
q_enqueue
f_write_buffers
pk_get_narrivals
er_abort
pk_get_first_arrival
pk_get_last_arrival
st_get_station
pk_get_next_arrival
deter3x3
ll_to_latlon
pk_get_magnitudes
pk_get_first_magnitude
pk_get_next_magnitude
dequeue
xy_to_polar
unfold


fr_continue_freerun
quit
r_check_time
dsp_domain_check
t_trigger_check
o_ring_bell
da_get_current_time
dt_stop_ADC
s_reset_screen
da_reset
dt_get_channel_size
o_write_logfile
da_get_current_time
do_chan_fft
combine_chan_bands
update_activity_counter
check_chan_spectra
check_chan_calib
s_display_triggers
display_bands
da_get_first_buffer
trigger_window_check
triggered_on_channel
er_abort
q_insert_link
q_enqueue
da_get_next_buffer
s_display_triggers
q_dequeue
QuitPwr
da_get_head_buffer
da_get_next_buffer
dt_get_dc_offset
t_get_channel_trigger_state
q_delete_link
er_abort
compute_dc_offset
q_dequeue


h_update
s_display_bar
pk_pick_arrivals
post_trigger_analysis
set_stoptime
diff_time
dt_get_number_of_ext_buffers
build_storage_stat
display_bar
dt_get_scan_count
t_get_head_trigger
t_get_next_trigger
st_get_station
er_abort
suds_initialize_tag
suds_initialize
st_get_authority
u_timestamp
insert_arrival
t_display_triggers
o_open_logfile
u_get_diskspace
q_insert_link
q_enqueue
u_build_timeline