NOTES ON INTERNAL SOFTWARE DESIGN REVIEW MEETING OF 18 DECEMBER 1992

Larry R. D'Addario
6 January 1993

OVLBI-ES MEMO NO. 37

Attendees:
Ed Meinfelder, Doug Varney, Larry D'Addario, Bill Shillue (part time), Dave Burgess -- Green Bank
Ron Heald, Ken Sowinski -- Socorro, by telephone

This was a very specialized review of certain critical software design work done since the overal project CDR on October 15, 1992. It included the design of most internal data structures, the station initialization code, the satellite geometry calculations, and the schedule dispatcher.

The meeting followed fairly closely the agenda given in Appendix A. Most materials presented as handouts or viewgraphs are reproduced in Appendices B (Larry), C (Doug), D and E (Ed). (Appendices are included only with hard copies of this report.) Earlier drafts of these notes were reviewed by the participants. Comments and explanations added by them after the meeting are given in square brackets [...].

1. Development Plan and Schedule -- Larry

The overall structure of the control system was reviewed briefly, with reference to a block diagram of the environment (Fig. B-1) and the current top-level data flow diagram (B-2). The development plan was presented as a Gantt chart (B-3), leading to completion at the end of June 1993. It is expected that considerable testing, debugging, and revision will be needed in the second half of 1993. Work to be reviewed at this meeting is restricted to that scheduled to have been done since the CDR in October.

[Ken -- For the project as a whole, June of 1993 seems optimistic, but possible if it is clear to all what is needed by that date. I think the first two action items (see below) must be resolved for this to be true.]

2. Internal Data Structures -- Doug

A. Monitor-related data.

Doug presented a design in which the monitor schedule, log schedule, checking limits, and monitor data are combined into a single structure rather than being separate. This structure is organized first by module, then by MCB address within a module, then by anomalies presently associated with a particular address (see vg1, vg5, and Figure C-1 in Appendix C).

The following concerns were raised during discussion.

(a) Ron - Does the monitoring/logging schedule need to be dynamically changeable? [Reply by Larry: Probably not on short time scales, but the whole thing will be kept in the Station Parameters File, so different schedules could be loaded for different purposes, e.g., test mode vs. normal mode.]

(b) Ken - The plan to have the CHECK task and others access previously-acquired monitor data is like the VLA, but not like VLBA (which allows each task to communicate on the MCB independently). The plan to store monitor results as MCB data rather than in user units is also unlike the VLBA.

(c) Larry - It is not clear how the checking limits can be made to depend on the current station state. Also, the design does not allow for interaction between the values at different MCB addresses. (It was not clear during the meeting whether any need for the latter exists, but the former is quite critical.)

[Ken - Some examples of error checks that require examining more than one monitor point are: (1) For the VLA dewars, the quantity (He supply pressure - He return pressure) should be within certain limits. (2) Certain combinations of outdoor temperature and dewpoint are interesting enough for checker to warn about it, like DP=temp and temp.LE.0 deg C. (3) In VLA terms, you might be interested in some combination of Gated Total Power and Synch Detector voltages.]

(d) [Ken -- The real issue here is discussed under E below: whether to use the VLBA code or not. The answer to this answers the first two action items and clearly sets what has to be done.]

B. Station Parameters

Straightforward structures were presented for the recorder parameters, pointing correction parameters, and certain geometric parameters like station coordinates and zones of avoidance.

Discussion:

(a) Ron - What utilities will be used to maintain the recorder parameters? VLBA station software keeps copy of structure in binary file, with a screen provided for loading, saving, editing. VLBA correlator keeps data in ASCII file, using text editor to maintain. [Doug - Our intention is to keep these in the same form as the VLBA, so that the VLBA screen can be used to maintain them.]

C. Station State

Doug presented a design in which the state consists of file position pointers for each of the two input files; an array of integers representing the software state; and a linked-list of MCB commands giving the last command sent to each valid address (vg3, Figure C-2).

Discussion:

(a) Larry - It is not clear what would be recorded in the "flags" array.

(b) Larry - Which tasks will maintain this list? The DFD shows it being maintained by DISPATCHER and INITIALIZE only. It is important that the structure show the *desired* state of the station, which is not necessarily the actual state.

(c) Larry - Some important information about the desired state may not be expressible as values sent via the MCB, e.g. the overall station mode, the orbit file in use, etc. These ought to be added to the structure.

(d) Ed - The DISPATCHER should be able to specify the desired state at a higher level than MCB commands.

(e) Ron - The design is not at all like the VLBA "observ" structure, so it cannot be substituted for the latter in VLBA "set*" and "get*" routines; thus, it seems that the latter routines must all be re-written.

(f) [Ken - I almost wonder if it pays (in simplicity) to include the full state for each line of the schedule file? By "line" I mean a line in the ascii file representing the schedule. I am suggesting a schedule format that allows you to know the full station-state by examining one element in the schedule list. My impression is that it takes very little information to characterize the desired state of the station.] [Larry - I don't agree. The command ("schedule") file needs to be flexible enough to allow changing *parts* of the station, so we cannot specify the whole state on one line. Nevertheless, the point that the station state structure could be very simple may be valid.]

D. Access Rules

Doug plans that MONITOR will lock out access to the monitor structure during updates by use of a semaphore (vg6). LOG and STATUS will block on this semaphore, whereas CHECK will run only after being triggered by MONITOR.

Discussion:
(a) Ken - Is it really necessary to control access in this way? What harm will occur if, say, CHECK is running while MONITOR is updating a value?
(b) Ken - If locking is needed, there are two other possible methods: task priorities and time slicing. In the latter, each task runs during an assigned interval so that access conflicts can't occur.
(c) Larry - What about the station state structure? For it, an even more complicated specification is needed for exactly who can modify it.

E. VLBA Code Affected

Doug presented a list of VLBA data structures and the VLBA C-functions that depend on them (vg7). It appeared that nearly all of this VLBA code would not be used in the present design. The idea of the new design is to keep the executable code for monitoring and station state maintenance completely generic (i.e., it does the same thing for all modules and monitor points, and need not have detailed knowledge of how anything works) and therefore simple, with all necessary information about particular devices being kept in the data structures. Thus, although large amounts of VLBA code would not be used, the amount of new code required would be small. In spite of this, Doug pointed out that most of the VLBA functions could still be retained unchanged. [The idea here is that it would allow several important "screen" programs to be kept for operator interfaces, using the corresponding "get*" routines, even if the automatic system operates completely differently and independently.]

Discussion:
(a) Larry - The design is more clean and elegant than that used at the VLBA, but we have not really shown that the generic organization and processing will be flexible enough for all the hardware.
(b) Ron - Although it is admitted that the VLBA approach is in some ways awkward, it has the advantage that it already exists and is working.
(c) [Ken - I would suggest extending this so that MONITOR consists of a collection of chk*() calls. Then CHECK is a byproduct of MONITOR. The caller of the chk*() routines would then save the stuff returned in any structure you choose for use by LOG, STATUS, etc.] [Larry - CHECK could be a subprogram of MONITOR, as Doug has planned, but not the other way around, as Ken suggests. We want the flexibility to monitor and log things for which there is not presently a checking algorithm.]

3. Initialization Task Design -- Doug

The design involves copying station geometrical parameters, recorder parameters, and pointing correction parameters from the Station Parameters file to the appropriate structures; then copying a list of MCB commands from the file to the station state structure; then sending those MCB commands from the station state structure to the hardware (vg2, vg2a).

Discussion:

(a) Larry - As noted earlier, it is not clear whether a set of MCB commands can properly express the desired station state, so it may also not be able to specify how to initialize the station.

(b) Larry - There needs to be a provision for recovery after power failure. This needs a carefully crafted algorithm (see also discussion under Schedule Dispatcher, below).

(c) [Larry - How do we know that the initialization was successful? How are errors reported?]

(d) [Larry - Initialization of the addresses of MCB interfaces has been omitted.]

4. Schedule Dispatcher Design -- Ed

The design is described in Appendix D. The schedule file [which should be called the "command file" in all our documentation from now on, to distinguish it from the externally-supplied schedule] is parsed according to a defined syntax, and then individual functions are called to implement each command. The design allows for macros to be defined and called, and for other command files to be referenced for inclusion. The main use of the latter would be for the definition of standard macros. The design includes provisions for special checks for restart procedures when it is first activated.

Discussion:

(a) There was considerable discussion of the procedures for restarting after a power failure. [Actually, this should be handled within the INITIALIZE task rather than here.] The plan (from the purple book) was to keep following the command file as long as UPS power to the computer is retained, even though other hardware is off; then to checkpoint the station state and file pointer just before computer power is lost. Upon restart, station state and file pointer are restored (if necessary), then the hardware is re-set into the specified state, then (if necessary) the command file contents are executed from the pointer to the present time.

Ken - Why not forget the checkpointing, just redo everything from the beginning of file?

Larry - Maybe, but BOF may not be the beginning of a run, so the state may not be well defined; it would be better to redo from the last RESET, which might be BOF or might not.

We agreed to study this, and to implement whatever is simplest, with the goal of achieving automatic recovery without operator intervention.

(b) Larry - Which commands are primitive and which are implemented as macros? In many cases, it could be done either way, but some things *must* be primitive, and it would be good to have a complete list of these.

(c) Ron - This seems to combine the functions of "ldsked" and "newd" in the VLBA system.

5. Geometry Calculations -- Ed

The design is described in Appendix E. Basically, a subset of the JPL NAIF/SPICELIB software package will be used to read and evaluate the satellite ephemeris file in real time. The satellite position and velocity will then be transformed to local coordinates of date using VLBA routines for precession, nutation and sidereal time, plus knowledge of the geographical station location. This is then transformed to the link delays and the pointing angles. Actually, the link delays are needed for two slightly different positions at any given earth station time, corresponding to the uplink and downlink, and these must be found iteratively. Tests show that two iterations are almost always enough, and that an accurate set of these calculations should take only a few milliseconds. Furthermore, cubic spline interpolation (in the two-way timing hardware) seems to be good enough that the precise calculations are needed only every 10 sec or so.

For extracting the satellite ephemeris, the plan is to port a small portion of the JPL code from FORTRAN to C and to run it in real time under VxWorks.

Discussion:

(a) Ken -- Is it necessary to include corrections for polar motion? Is it necessary to include diurnal abberation?

[I have looked into this a bit. The magnitude of polar motion is only 9 meters. If you are happy lumping this in to a general "station position error" than it need not be considered. The diurnal aberration correction is not needed to point the antenna. I think I have convinced myself that you do not need to consider this for the distance calculation, so it too is not needed.]

(b) Ken -- The plan to port some of NAIF from an interactive to a real time environment, and from FORTRAN to C, seems similar to the VLBA correlator's use of CALC. It might be valuable to learn the details of how they did it.

(c) Ken -- If the precise calculations are needed only every 5 to 10 sec, then perhaps they can be done off-line, avoiding the problem of porting SPICELIB to the real time system. [If these calculations are done in real time, then consider splitting them into two tasks: a slow one to run every 10 sec, and a fast one to run at 16Hz (?) for interpolation.]

ACTION ITEMS:

1. Determine the additional effort required to implement code to support the newly designed data structures. Note that the present development plan assumes use of VLBA code for control of VLBA-type devices, including recorder and formatter. Exactly which VLBA code would need to be re-written? --Doug

2. Determine whether the Station State structure could be modeled after the VLBA "observ" structure, and if so whether it could be close enough to allow retaining of most hardware-specific VLBA code. --Doug

3. Create detailed plan for hardware initialization, including default states of everything. --Doug

4. Decide algorithm for restart after power failure. --Doug, Larry, Ed

5. Make a list of DISPATCHER commands that must be implemented as primitives. --Ed

6. Study the VLBA correlator's porting of CALC to a real-time environment, since there may be some useful parallels to our porting of a portion of NAIF/SPICELIB. --Ed, Larry

7. Determine whether polar motion and/or diurnal abberation needs to be included in the geometry calculations. --Larry

APPENDIX A

OVLBI Earth Station Project

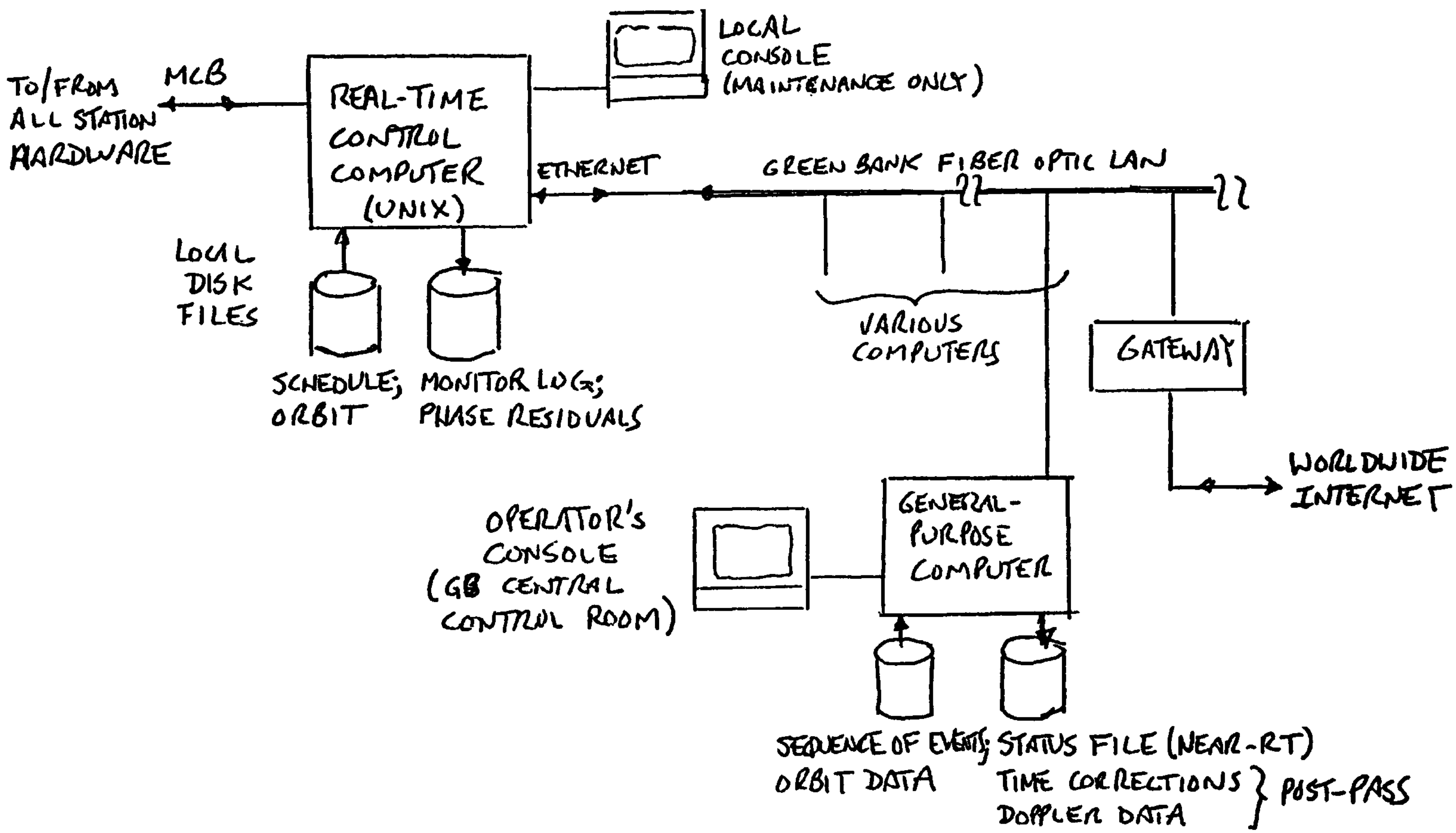
Software Design Review Meeting, December 18, 1992

~~PRELIMINARY~~ AGENDA:

(Times, in EST, are approximate and include discussion.)

1100-1110	Software Development Plan and Schedule	L. D'Addario
1110-1150	Internal Data Structures Design A. Description of structure contents: Station state Monitor data Monitor schedule Logging schedule Monitor limits Poining parametrs Recorder parameters B. Rules for modifying and accessing by tasks C. Implications for VLBA code porting: --VLBA modules affected --modifications required	D. Varney
1150-1215	Initialization task design	D. Varney
1215-1300	--- Lunch Break ---	
1300-1330	Geometry calculations A. Satellite orbit processing: NAIF routines --description --timing tests --accuracy tests B. ES location: precession, etc. C. Offline software design [if any] D. Realtime software design	E. Meinfelder
1330-1400	Schedule dispatcher design A. Command file syntax B. Initial command set description C. Realtime dispatcher task design	E. Meinfelder

GREEN BANK OVLBI EARTH STATION: MONITOR AND CONTROL OVERVIEW



APPENDIX B
Fig B-1

- LRD 920319

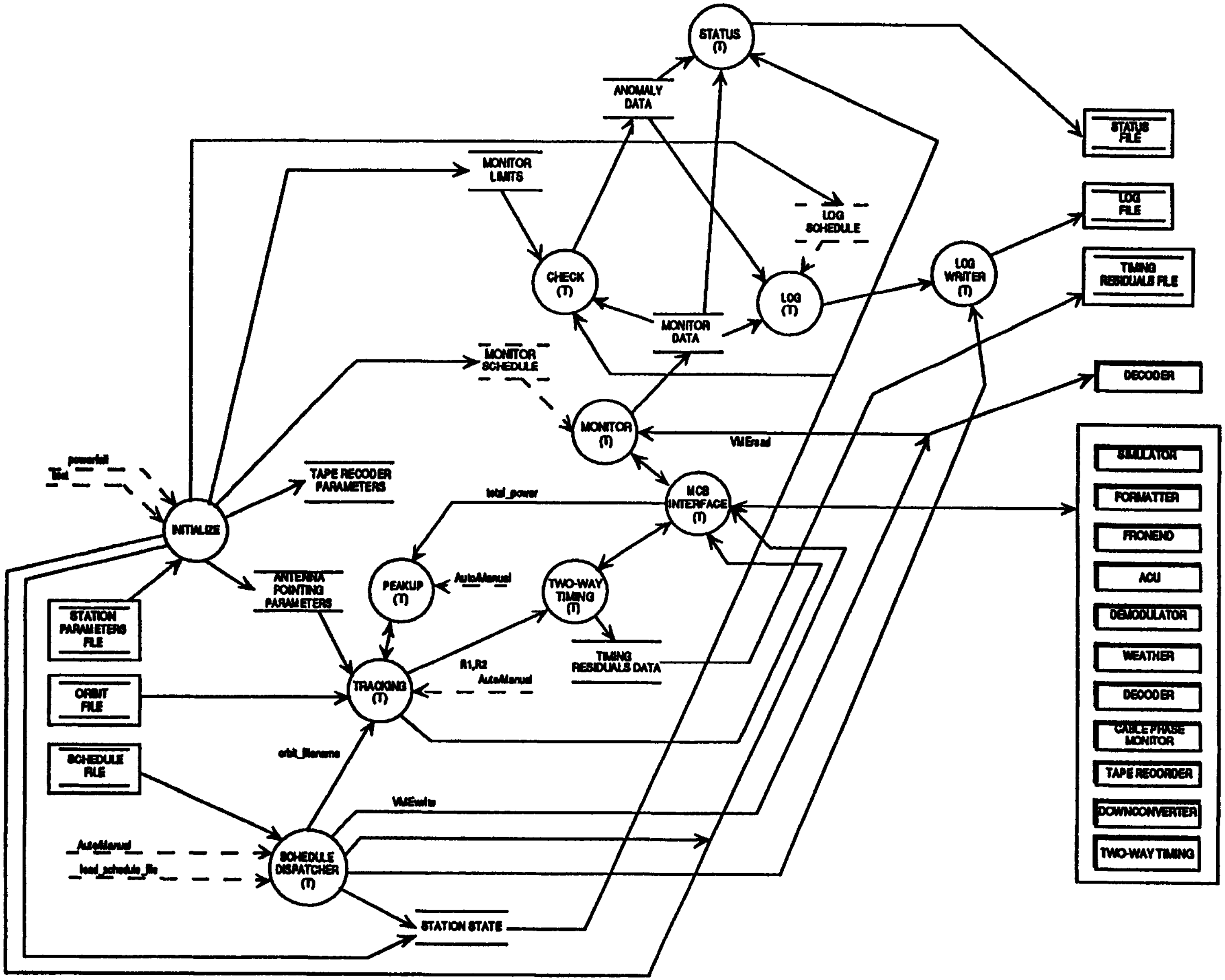
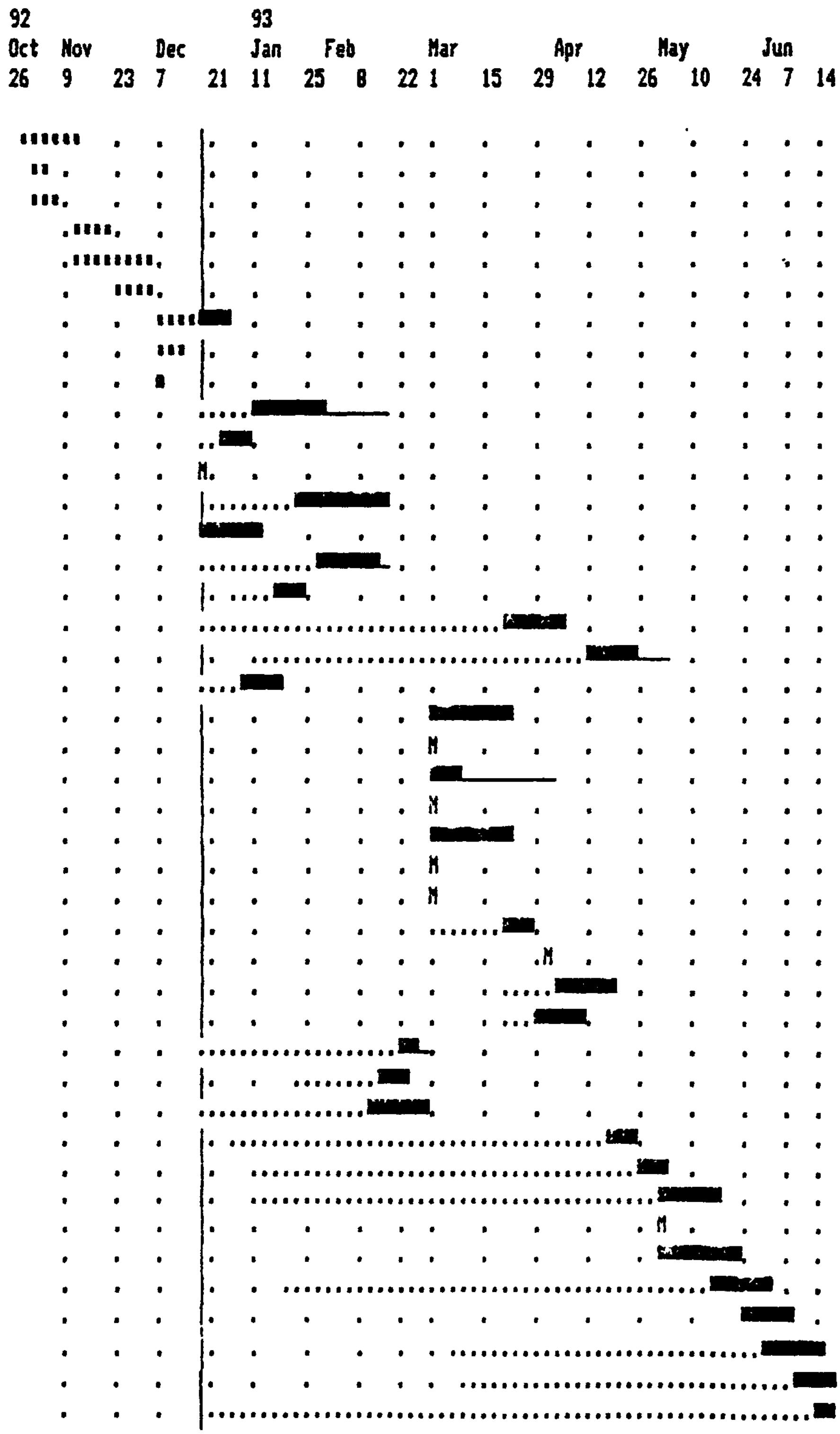

STATUS (T)
ANOMALY DATA
STATUS FILE
MONITOR LIMITS
LOG SCHEDULE
LOG FILE
CHECK (T)
LOG (T)
LOG WRITER (T)
TIMING RESIDUALS FILE
MONITOR DATA
MONITOR SCHEDULE
DECODER
MONITOR (T)
VMEread
powerfail
TAPE RECODER PARAMETERS
MCS INTERFACE (T)
total_power
SIMULATOR
FORMATTER
FRONEND
ACU
DEMODULATOR
WEATHER
DECODER
CABLE PHASE MONITOR
TAPE RECORDER
DOWNCONVERTER
TWO-WAY TIMING
INITIALIZE
ANTENNA POINTING PARAMETERS
PEAKUP (T)
AutoManual
TWO-WAY TIMING (T)
STATION PARAMETERS FILE
TIMING RESIDUALS DATA
R1,R2
AutoManual
ORBIT FILE
TRACKING (T)
SCHEDULE FILE
orbit_filename
VMEwrite
AutoManual
load_schedule_file
SCHEDULE DISPATCHER (T)
STATION STATE

Fig B-2

Schedule Name : Software -- GB earth station
Responsible : D'Addario
As-of Date : 18-Dec-92 Schedule File : SOFTWARE

Fig B-3

Task Name	Resrc	Stat	End Date
NAIF tests	EM		12-Nov-92
Organize code	DV		4-Nov-92
DSP conference	DV		9-Nov-92
Finish ACU screens	DV		20-Nov-92
Geometry/D	EM		7-Dec-92
Data structures	DV	C	4-Dec-92
Initialize Task/D	DV	C	23-Dec-92
Dispatcher v1.0/D	EM	C	14-Dec-92
Decide orbit calc metho	EM, LD		7-Dec-92
Dispatcher v1.0/I	EM	r	28-Jan-93
Monitor Task/D	DV	C r	8-Jan-93
Design Review #1	LD	C	16-Dec-92
Initialize/I	DV	r	12-Feb-93
Decoder Screen/D	EM		12-Jan-93
Geometry-tmp/I	EM	r	11-Feb-93
Check Task/D	DV	C r	22-Jan-93
Clock task	DV	C r	5-Apr-93
Dispatcher full/I	EM	r	26-Apr-93
Pointing Task/D	DV	C r	15-Jan-93
Doppler pgm/D	EM		22-Mar-93
TWT protocol specified	LD	C	1-Mar-93
TWT Task/D	LD		8-Mar-93
DeltaT interface	LD	C	1-Mar-93
DeltaT pgm/D	DV	C	22-Mar-93
Doppler interface	LD	C	1-Mar-93
Decoder protocol specif	RH	C	2-Mar-93
Decoder Control/D	EM	r	29-Mar-93
Software Review #2		C	1-Apr-93
DeltaT pgm/I	DV	C r	19-Apr-93
Doppler pgm/I	EM	r	12-Apr-93
TWT Screen/D	DV, LD	r	25-Feb-93
Pointing Task/I	DV	r	22-Feb-93
Geometry/I	EM	r	26-Feb-93
Monitor Task/I	DV	C r	26-Apr-93
Log Tasks/D	DV	C r	3-May-93
Check Task/I	DV	C r	17-May-93
Sched interface	LD	C	3-May-93
Sched converter/D	EM	C	24-May-93
Log Tasks/I	DV	C r	1-Jun-93
Sched converter/I	EM	C	8-Jun-93
TWT Task/I	DV	C r	15-Jun-93
Decoder/I	EM	C r	22-Jun-93
TWT Screen/I	DV	C r	22-Jun-93



Detail Task ===== Summary Task ***** Baseline
(Progress) ===== (Progress) >>> Conflict
(Slack) ===__ (Slack) .. Resource delay
Progress shows Percent Achieved on Actual M Milestone
----------------- Scale: 2 days per character -----------------

TIME LINE Gantt Chart Report, Strip 1

APPENDIX C

```
ViewGraph :: MONITOR DEPENDENT DATA --------------------------------------------

#define NULL            0
#define VALID_ID        1
#define ANALOG          0
#define DIGITAL         1

#define ACU_ID          0x00            /* antenna control unit */
#define X_RCVR_ID       0x05            /* X-band receiver */
#define Ku_RCVR_ID      0x07            /* Ku-band receiver */
#define RdTripMON_ID    0x15            /* Round Trip cable phase monitor */
#define Maser_ID        0x16            /* H-maser */
#define Weather_ID      0x18            /* weather instruments (VLBA address) */
#define Synth1_ID       0xF0            /* ES synthsizer : 7-16 GHz */
#define Synth2_ID       0xF1            /*               : 8.6-9.4 GHz */
#define Recorder1_ID    0x2A            /* tape recorder 1 */
#define Recorder2_ID    0x3A            /*               2 */
#define TWT_ID          0xF2            /* two way timing unit */
#define FORMATTER1_ID   0x2C            /* formatter 1 */
#define FORMATTER2_ID   0x3C            /*           2 */

#define MPts              13            /* number of monitor points */
#define MaxPts          0x7F            /* maximum available h/w units */

--------------------------------------------------------------------------------
```

ID_list :: 'enumerated' list for hardware ids, used in monitor_set record

Devpts :: each device on the MCB has a number of monitor points, specified in Devpts, for each device

Msked :: the monitoring schedule can be different for each monitor point within a device. the data in Msked is for all points on the MCB that will be monitored, and is loaded into the general list.

Lsked :: retains the logging interval for each monitor point specified in Msked

Llimits :: all monitor points have a window of acceptability. the Llimits are the lower bounds for a given point that returns an analog value.

Ulimits :: is the upper bounds for the analog limits

Ptype :: contains the monitor point type, (analog:0)|(digital:1)

```
ViewGraph :: INITIALIZATION TASK PDL#1 ----------------------------------------

TASK Init                                -- initialize the station, start tasks

task_list = {SCHEDULE_DISPATCHER,
             SATELLITE_TRACKING,
             MONITOR
             LOG,
             STATUS,
             TWO_WAY_TIMING}

 SEQ ::
        counter IS 0
        semXCreate Sem_1sec, Sem_16persec

        LOOP
          IF counter <> MAX_IDS
            CALL fill_station_parms
          OTHERWISE
            EXIT
        END_LOOP

        CALL fill_station_monitoring_pts
        CALL setup_hardware

        CALL spawn_tasks(task_list)

END_TASK Init
```

ViewGraph :: INITIALIZATION TASK PDL#2 --

```
FUNCTION fill_station_monitoring_pts     -- fill the monitor/log structure
                                         -- req : ^monitor_record, index i
 SEQ ::
        offset_address IS offset.i
        monitor_time IS m_sked.i
        log_time IS l_sked.i

        ON first_in_list
          number_monitor_points IS device_points.i

        IF analog
          type IS #analog
          upper_bounds IS upper.i
          lower_bounds IS lower.i
        OTHERWISE
          type IS #digital
          bit_map IS bits.i

        ^error IS #NULL

        RETURN monitor_record

END_FUNCTION fill_station_monitoring_pts


FUNCTION read_file_into_parameters       -- read the station params into mem

 SEQ ::
        LOOP
          READ file(station_parameters)
           WRITE TO_tape_recorder_parameters_record
           WRITE TO_antenna_pointing_paramters_record
           WRITE TO_station_state

          IF EOF
            EXIT
        END_LOOP

END_FUNCTION read_file_into_parameters

FUNCTION setup_hardware                  -- initial hardware setup

 SEQ ::
        LOOP
          FROM station_state
          MCB(ID,command_from_station_state)

          IF EOS
            EXIT
        END_LOOP
```

ViewGraph :: STATION STATE STRUCTURE --

```
typedef struct rel_state
{
        int     relative_address,       /* offset from ID                       */
                commanded_value;        /* last command sent to the device      */

        struct  hardware_state *next;   /* next address and its 'state'         */

} hardware_state;


typedef struct state
{
        int     sked_filePosition,      /* event rec pos in schedule file       */
                orbit_filePosition;     /* event rec pos in orbit file          */

        int     flags[];                /* flag settings                        */
        struct  hardware_state  *list;  /* list for current hardware settings */

} station_state;
```

ViewGraph :: MONITOR|CHECK|LOG SETTUP ---

```
int Msked[] = {1,4,                                          /* ACU_ID          */
               5,                                            /* X_RCVR_ID       */
               6,2,4,12,                                     /* Ku_RCVR_ID      */
               4,5,6,2,6,8,8,                                /* RdTripMON_ID    */
               12,13,1,4,4,6,8,5,6,4,2,7,                    /* Maser_ID        */
               1,3,2,                                        /* Weather_ID      */
               4,10,10,12,10,4,5,6,7,10,3,6,4,3,5,7,         /* Synth1_ID       */
               5,3,1,                                        /* Synth1_ID       */
               9,8,7,11,10,5,                                /* Recorder1_ID    */
               2,6,1,8,7,2,7,4,7,                            /* Recorder2_ID    */
               1,                                            /* TWT_ID          */
               1,7,2,4,2,7,6,                                /* FORMATTER1_ID */
               10,9,2,4                                      /* FORMATTER2_ID */
              };

int Lsked[] = {1,4,
               6,
               6,6,6,6,
               2,2,2,3,4,4,2,
               4,4,4,4,4,5,5,5,5,5,5,4,
               5,2,4,
               5,6,6,3,6,5,3,6,3,2,4,4,5,6,3,3,
               1,3,4,
               4,4,4,3,2,3,
               4,4,5,5,5,3,3,3,4,
               1,
               4,4,4,5,5,5,2,
               5,7,8,9
              };

int RelAd[] = {1,2,
               0,
               0,2,3,5,
               1,3,4,6,7,8,10,
               2,3,4,6,7,8,9,10,12,13,14,16,
               0,1,2,
               0,2,3,4,6,7,8,9,10,11,12,13,15,16,17,19,
               0,1,2,
               0,1,2,3,5,7,
               1,2,3,4,5,6,7,8,9,
               1,
               0,4,5,6,7,8,9,
               1,2,3,4
              };
```

ViewGraph :: MONITOR DATA STRUCTURE --

```
typedef struct error
{
        long    time_of_first_occurance;
        int     error_value;
        struct  device_error *next;

} device_error;

typedef struct spec
{
        int     NpointsPerDevice,        /* number addressable points, this ID */
                MonitorPointType,        /* what is type of each point, A|D?   */
                relative_address,        /* relative address offset from ID    */
                monitor_schedule,        /* when to sample each point          */
                log_schedule;            /* and when to log it                 */

        float   upper_limits,            /* monitor limits .. upper            */
                lower_limits;            /*                .. lower            */

        long    bits;                    /*         valid bit string           */

        struct  device_error *errnext;   /* ptr to next anomaly record         */
        struct  device_spec *next;       /* ........... monitor point rec      */

} device_spec;

static struct
{
        int             device[MaxPts]; /* this point have an ID? (0:N,1:Y)   */
        device_spec     *p[MPts];       /* ptr to device spec record          */

} monitor_set;
```

```
ViewGraph :: MONITOR DATA R/W ACCESS -----------------------------------------------

        MONITOR TASK    -- needs semTake, semGive

        CHECK TASK      -- can access after MONITOR which provides trigger

                                1. needs to have no updates in progress

        LOG TASK        -- will require that no updates be in progress

                                1. can make a copy or
                                2. use semaphores

        STATUS          -- same as log

        concerns        -- 1. no access during read-modify-write
                           2. no read-write update
                           3. no write update
```

ViewGraph :: VLBA AFFECTED MODULES & DATA ------------------------------------

```
MONITOR DATA RECORD (monData)    {loger, wrlog}

OBS BLOCK           (monit1)     {wrlog}

FLAGER              (flager)     {flager, loger, wrlog}

CHECK               (checkfixed) {wrlog}

                    (checker)    {checker, getfe, wrlog}

                    (checkmsg)   {acu2, acuerr2, check, checkall, checker,
                                  chkacustat2, chkclk, chkfmtstat, chkfrstat,
                                  chkmic, chkrecstat, fmterr, fr, getacu2,
                                  getcrut, cetfe, cetfmt, cdtfr, getmas,
                                  getrec, getftm, getsw, getsyn, getwea,
                                  recerr, setall, wrlog}

TAPE                (tapevsn)    {wrlog}

WEATHER             (weather)    {check, chkmic, getwea, logboot, loger,
                                  precip, rcmdd, screen, setrefr, setwea,
                                  sn, wea, weamenu, weapwr, wrlog}

EQUIPMENT           (equipm)     {cfixEquipment, equisave, fecheck, recparm,
                                  sta, usedisk, wrlog}
```

FIGURE B-1

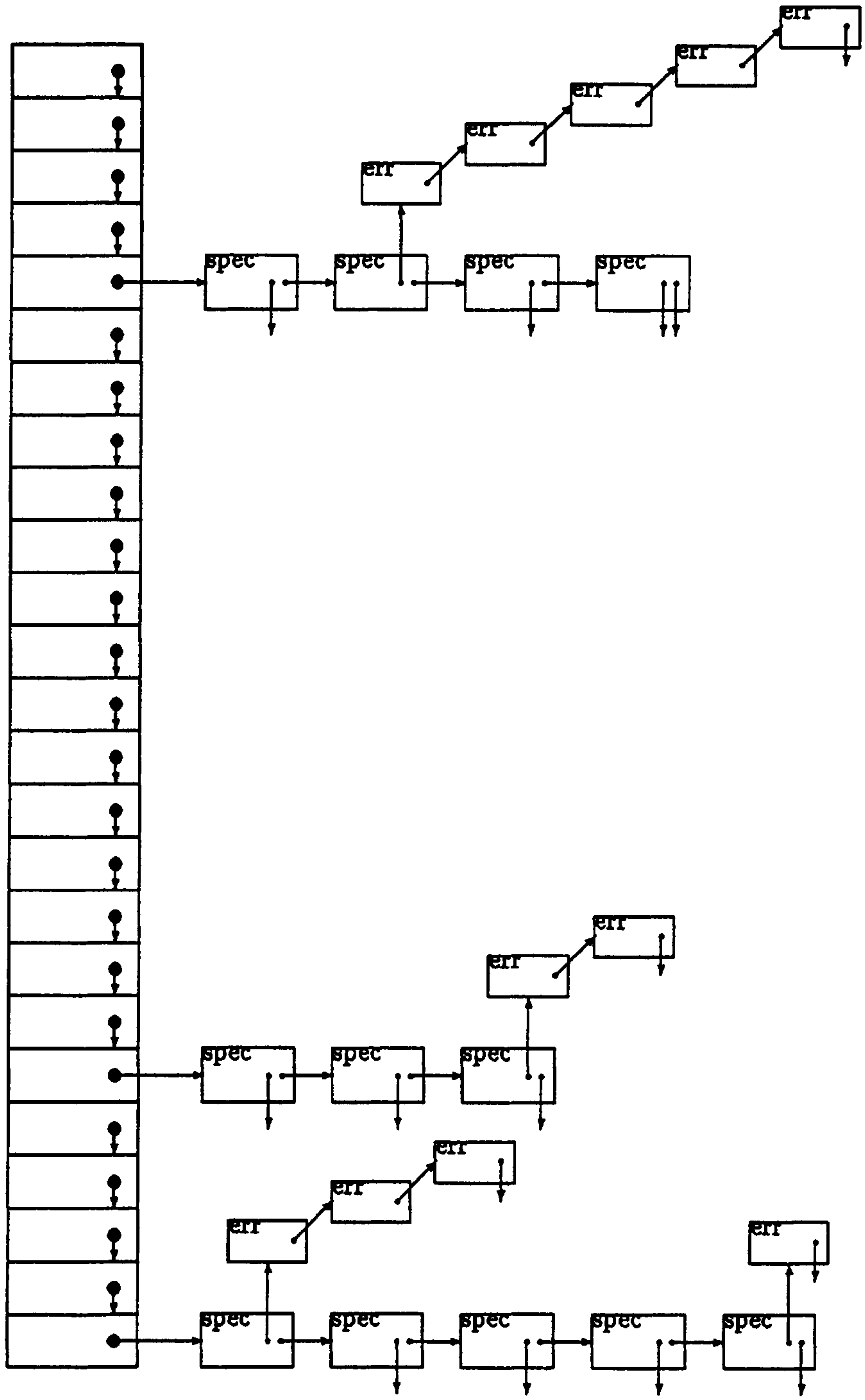
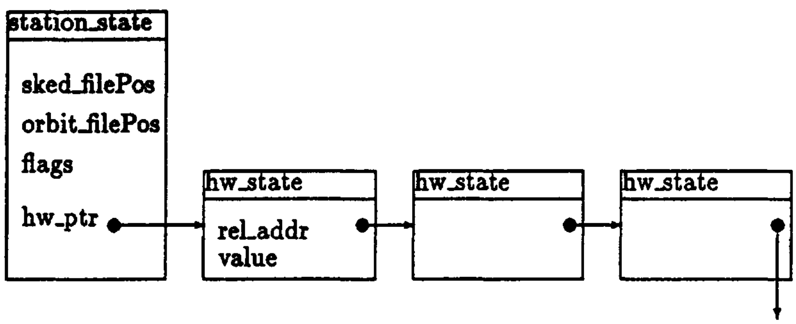

spec
spec
spec
spec
err
err
err
err
err
spec
spec
spec
err
err
spec
spec
spec
spec
spec
err
err
err
err

FIGURE B-2


station_state
sked_filePos
orbit_filePos
flags
hw_ptr
hw_state
rel_addr
value
hw_state
hw_state

APPENDIX D

II. Schedule Parser

A. Explanation of the Schedule Parser

The schedule parser is the driver for the user interface of the Earth Station. The user writes command in a file called a schedule file. The Schedule File's first command must be time stamped with a UTC time. All following records may or may not have an associated time. This associated time is the time at which the command must be executed. All following commands that do not have an associated time are assumed to be executed at the last associated time.

Now commands are read in one at a time, the time is checked. if the time has not yet come to pass, then the schedule parser will wait until that time comes about.

B. Data Structures

Two major data structures are used to implement to features of the Schedule Parser.

The first is the defines table. The defines table stores static definitions. Any parameter that begins with an alphabetic character where an numeric character was expected, will be looked up in the defines table, if not found and error will be reported.

Defines table structure:

```
struct def_tab_node {
     char                      symbol[13];
     double                    value;
     struct def_tab_node       *next;
};

struct def_tab_node      *define_table[26];
```

The second data structure is the Macro Table. The Macro Table stores the text to be executed upon invocation of the macro. All commands that do not match any known command are checked to see if they are macros. Macros can be a list of commands of any number. Macro table structure:

```
struct macro_txt_node {
     char                      *text;
     struct  macro_txt_node    *next;
}

struct macro_tab_node {
     char                      symbol[13];
     struct macro_txt_node     *next;
}

struct macro_tab_node    *macro_table[26];
```

C. Startup

Upon startup the Schedule parser checks for existence of backup state file that will determine what the next action is performed. Is this needed, are any functions the *MUST* be performed in sequence? on the level of the schedule parser? As it will only have access to the current schedule file. If it has access to multiple schedule files then this implies that the Earth Station is either processing the file or retrieving it. Either function is silly for a real time computer to be performing. This is true if we have other computers to perform the tasks available. I just don't know anymore. A RESTART_NO may be needed to discard previously performed actions.

D. Command Syntax

[<time>] COMMAND [<parameters> [...]]

In the above example the time is shown as optional. This is true for all cases except the first record. The first record must have a time associated with it. If the first record fails this criteria then an error will be reported and the pass aborted.

<macro name>	will execute a macro command.
ACQUIRE <timeout>	Begin to attempt to automatically acquire downlink signals from the satellite. Includes execution of PEAKUP. Once adequate signal is being received, executes tracking pass initialization sequence. If not accomplished in timeout seconds, send alarm to the operator.
AUTOTEST	Performs station test sequences
CALTONEOFF	End detection of test tones; write results to a log.
CALTONEOFF	End detection of test tones; write results to a log.
DATAMODE <TYPE>	Sets up the demodulator and decoder for the satellite data datamode. Types are RA0-2, VSOP, & TEST.
FORMATTER	Setup the formatter
LOAD=<file name>	will load in an include file consisting of any number of commands.
MACRO=<macro name>	will read in a macro command and store it for later use.

MCB <address>=<value>	will write a datavalue to a specific address on the MCB
PEAKUP	Causes antenna pointing to be scanned around the nominal position in order to peak up on the satellite signal. Computes the pointing error and applies appropriate pointing correction until next PEAKUP or RESET
RESET	will reset the Earth Station w/ defaults
RFMODE (RA¦VSOP¦TEST)	Set switches in receiver and transmitter to Radioastron or VSOP or test.
STANDBY	will stop antenna at present position and set the brakes
STARTUP	Turn on power to antenna servo and cause computer to take control. Ensure that the brakes are set.
STOW	will place the antenna in the stow position and set the brakes, tracking computations will still continue
TAPE <parm>=<value>	Sets recording headstack to given position enable specified head groups for writing. Starts tape motion at a given speed, if speed is <= 0 tape direction is reversed. Settable parameters include speed, head position, and wenable.
TRACK <orbit file>	Antenna must have been given a STARTUP command prior to this command. Code will be executed that releases the antenna brakes.
TRANSMITTER <power>	Turn on the transmitter at a specified power level. Frequency should have been set up earlier. Zero or negative turns xmtr off.
TWTMODE (RA¦VSOP¦TEST)	will set the two-way timing mode to the correct value

E. Real Time Software Design

```
main routine  {
    open schedule.file

     /* check and see if we stopped in the middle */

    if (get_num_read_done() > 0)
        read get_name_read_done() records from schedule.file

     /* now parse the schedule file */

    do_parse_file(schedule.file)
}


/*   do_parse_file() is the main loop that will process the
     schedule file, one record at a time
*/

do_parse_file(FILE)  {

     /* while there are still records to be read from FILE */

    While (FILE)  {
        read a RECORD from FILE

        remove white space from record
        get time for command

          if (no time and first record)  {
               log error
               quit
          }
        while (time of command < current time)
            wait on 1HZ semaphore (signaled by timer process)

          /*   parse that command and execute it, checking return
               value if there was an error log that error
          */

        if ((error = command_parse_exec(cur_token)) < 0)  {
            sprintf(err_buf,"on line %i in the file %s\n",linenum
                ,SCHEDULE_FILE);
            error_message(error,err_buf);
        }
    }
}
```

```
/*

do_reset()  will reset all default station parameters

syntax:
    [<time>] RESET
*/

do_reset()
{
     log command action
     get default station parameters from default file
     send out default parameters for TAPE DRIVES via MCB
     send out default parameters for SYNTHESIZERS via MCB
     send out default parameters for DEMODULATOR via backplane
     send out default parameters for TWT via MCB
     send out default parameters for DECODER via MCB
     send out default parameters for ANTENNA via MCB
}

/*

do_load()  will load in an include file consisting of any number of
commands.

syntax:
    [<time>] LOAD=<file name>
*/

do_load()
{
    parse out filename argument
    load_include(FILENAME);
}

load_include(FILENAME)
{
    open (FILENAME)
    do_parse_file(FILENAME)
}
```

```
/*
do_macro()  will execute a macro command.

syntax:
    [<time>] <macro name>
*/

do_macro(MACRO_NAME)
{
    lookup macro name in macro table
    if (no such macro)
        return an ERROR
    retrieve MACRO BUFFER from macro table
    mb_ptr points to MACRO BUFFER
    l_ptr = line_buffer

    while (*mb_prt != '\0')  {
        while   (*mb_prt != ('\0' OR '\n'))
            l_ptr++ = *mb_prt++;
        get command cur_token from line_buffer

        /*      parse that command and execute it, checking return
                value if there was an error log that error */

        if ((error = command_parse_exec(cur_token)) < 0)  {
            sprintf(err_buf,"on line %i in the file %s\n",linenum
                ,SCHEDULE_FILE);
            error_message(error,err_buf);
        }
    }
    log action
}
```

```
/*
do_readmacro()  will read in a macro command and store it for later
use.

syntax:
     [<time>] MACRO=<macro name>
          <macro text>
          ...
     END MACRO
*/

do_readmacro()
{
     parse out macro name
     allocate an entry in the MACRO table
     read a record of CURRENT FILE
     while (record != "END MACRO")  {
          copy buffer to new record, hash_loc(macro name) in table
          read CURRENT FILE record
     }
     log action
}

/*
do_twtmode()  will set the two-way timing mode to the correct value

syntax:
     [<time>] TWTMODE=(RA¦VSOP¦TEST)
*/

do_twtmode()
{
     parse out mode argument
     if (mode == "RA")
          do whatever it takes;
     else if (mode == VSOP)
          again do what ever it takes
     else
          set test mode default parms
     log action
}
```

```
/*

do_autotest()  Performs station test sequences

syntax:
    [<time>] AUTOTEST
*/

do_autotest()
{
    /* test diagnostics are as yet unspecified */
     force checker to check all values now and report
     log action and result
}

/*

do_standby() will stop antenna at present position and set the
brakes

syntax:
    [<time>] STANDBY
*/

do_standby()
{
    acu_BRAKES_ON();
     disable satellite geometry module from sending out any
     pointing commands
    log action
}

/*

do_stow() will place the antenna in the stow position and set the
brakes,
    tracking computations will still continue

syntax:
    [<time>] STOW
 */


do_stow()
{
     acuSTOW();
}
```

```
/*
do_acquire() Beg to attempt to automatically acquire downlink
signals from the satellite. Includes execution of PEAKUP.  Once
adequate signal is being received, executes tracking pass
initialization sequence. If not accomplished in timeout seconds,
send alarm to the operator.

syntax:
    [<time>] ACQUIRE <timeout>
*/

do_acquire()
{
    parse out TIMEOUT
    convert to a time
    fork off satellite geometry module
    fork off acquire_signal(TIMEOUT)
    log action
}

acquire_signal(TIMEOUT)
{
    signal.acquired = FALSE
    while (!signal.acquired)
        do_peakup()
    log action
}

/*

do_peakup() Causes antenna pointing to be scanned around the
nominal position in order to peak up on the satellite signal.
Computes the pointing error and applies appropriate pointing
correction until next PEAKUP or RESET

syntax:
    [<time>] PEAKUP
*/

do_peakup()
{
    acuPEAKUP()
    If (the signal was acquired)
        signal.acquired = TRUE
    log action
}
```

```
/*

do_mcb() will write a datavalue to a specific address on the MCB

syntax:
    [<time>] MCB <address>=<value>
*/

do_mcb()
{
    parse out ADDRESS
    parse out VALUE
    set up and MCB.message
    mcbio(MCB.message, sizeof(MCB.message))
    log action
}

/*

do_rfmode() Set switches in receiver and transmitter to Radioastron
    or VSOP or test.  Not too complex.

syntax:
    [<time>] RFMODE (RA¦VSOP¦TEST)
*/

do_rfmode()
{
    parse out MODE
    set receiver via MCB to MODE
    set transmitter via MCB to MODE
    update checker's structure that tracks RF MODE settings
    log action
}

/*

do_datamode() Sets up the demodulator and decoder for the satellite
data datamode

syntax:
    [<time>] DATAMODE (RA0¦RA1¦RA2¦VSOP¦TEST)
*/

do_datamode()
{
    parse out DATAMODE
    set demodulator to DATAMODE
    set decoder to DATAMODE
    update checker's structure that tracks DATAMODE settings
    log action
}
```

```
/*
do_transmitter() Turn on the transmitter at a specified power
level.
     Frequency should have been set up earlier. Zero or negative
     turns xmtr off.

syntax:
    [<time>] TRANSMITTER <power>
*/

do_transmitter()
{
    parse out power
    if (power < 0)
        turn off transmitter via MCB
    turn on transmitter at specified power via MCB
    log action
}

/*
do_caltonon() End detection of test tones; write results to a log.

syntax:
    [<time>] CALTONEOFF
*/

do_caltoneon()
{
    parse out all channel-frequency pairs
    write out value to the MCB to begin detection at for each pair
    log action
}


/*
do_caltoneoff() End detection of test tones; write results to a
log.

syntax:
    [<time>] CALTONEOFF
*/

do_caltoneoff()
{
    write out the correct values to the MCB to end detection
    log action
}
```

```
/*
do_startup()  Turn on power to antenna servo and cause computer to
    take control.  Ensure that the brakes are set.

syntax:
    [<time>] STARTUP
*/

do_startup()
{
     if (antenna.control != COMPUTER) return;
    acuSTARTUP()
    log action
}

/*
do_track() Antenna must have been given a STARTUP command prior to
this command.  Code will be executed that releases the antenna
brakes.

syntax:
    [<time>] TRACK <orbit file name>
*/

do_track()
{
     if (antenna.control != COMPUTER) return;
     if (startup_done == TRUE)
        exec brake release code by Doug Varney
        log action
     else
        log error
}

/*
do_tape()   Sets recording headstack to given position enable
specified head groups for writing.  Starts tape motion at a given
speed, if speed is <= 0 tape direction is reversed.

syntax:
    [<time>] TAPE SPEED=<ips> HEADPOSN=<microns> WENABLE=<abcd>
*/

do_tape()
{
    parse arg1name, arg1val
    parse arg2name, arg2val
    parse arg3name, arg3val
    get SPEED, HEADPOSN, WENABLE
    call init_com()              /* VLBA initial communications */
    call speed(unit, SPEED)      /* from VLBA code, set speed */
    call rec_out()               /* set headposn */
    call rec_out()               /* set WENABLE */
}
```

```
/*
do_formatter() Setup the formatter

syntax:
    [<time>] FORMATTER <?> [<?> ...]
*/

do_formatter()
{
    /* ? */
}
```

APPENDIX E

I. Satellite Geometry Software

A. Description

The satellite tracking software that runs on the Earth Station tracks the spacecraft by accessing ephemeris data supplied by JPL. The ephemeris data will be supplied in an SPK (Space Planet Kernel) file. The SPK format is used by SPICELIB, a FORTRAN library that is part of the software package called NAIF also supplied by JPL. NAIF, given ephemeris data, can perform many different types of calculations for the object of the ephemeris data. One such calculation is the position and velocity of the target body relative to another body.

The NAIF algorithms were not designed with real-time in mind. If the NAIF software package was used as is, it would have to be as offline software used before the pass producing a data file many times larger than the original ephemeris data file. For this reason, the iterative algorithms that determine the position and velocity of the spacecraft will be rewritten in C and performed in real time.

B. Timing Tests

Timing tests were performed with the NAIF software as shipped from JPL. The software was run "as is" on a SPARC IPC that had a "low" load average. A total of one thousand calls were made to SPKPV(), a SPICELIB routine that simply evaluates the ephemeris data for a given ephemeris time. The average time for a call to SPKPV() was 0.1240E-02 seconds.

C. Accuracy Tests

Cubic spline interpolation was performed over the entire orbit repeatedly to find the maximum interval at which approximation could be performed using a cubic spline within 1E-6 km (1mm). Based in the sample data from JPL, the interval was found to be 13 seconds. The implication is that if the sample data is representative of vsop's orbit then, when tracking the satellite, the NAIF evaluation algorithm for an SPK Modified Difference Array File's ephemeris data must be used at least once within a period of 13 seconds.

D. SPK File Format

File Header:

FILE ID:	'NAIF/DAF'	
ND:	2 Double Percision Components	
NI:	6 Integer Componenets	

Record Header:

ET1:	Begin time	(1st DP Component)
ET2:	End time	(2nd DP Component)
TARGET:	VSOP -71	(1st Int Compenent)
CENTER:	EARTH 399	(2nd Int Compenent)
IRF:	4	(3rd Int Compenent)
TYPE:	1 (fixed)	(4th Int Component)
BEGIN:	Begin Adr	(5th Int Component)
END:	End Adr	(6th Int Component)

Record Data:

a double percision array, 128 entries

E. Real Time Software design

Main Loop for Satellite Tracking

```
While (Tracking.On)  {
     wait on 16hz semaphore

     for downlink & uplink time:
          1- iteratively solve for time of Uplink/Downlink
          2- get geocentric XYZ position of Satellite
          3- get geocentric XYZ position of Earth Station
          4- calc Azimuth & Elevation
          5- add pointing corrections to Azimuth & Elevation
               (from last peakup and station parameters)

     /* point antenna to the spacecraft */
     acuMOVE_TO(elevation, azimuth)

     send Uplink, Downlink range & time to Two Way Timing
          (via pipe)
     log Uplink, Downlink range & time to log file.
}
```

PDL for each of the 5 steps:

1 - iteratively solve for time of Uplink/Downlink

```
get current ephemeris time
get light time of current range, and use that
     (+ for up/- for down) Tg
get position at Current time T (+/-) Tg
iterate until abs(T-Tg) < TOLERANCE
```

2- get geocentric XYZ position of Sat.

A - retrieve correct record from JPL SPK file by ephemeris time

```
/* create index of end times */
/* done once at startup */
read file_header
i = 0
while not eof()  {
     read record_header
     add 1 to i
     time_index[i] = record_header.end
     read record_data
     /* since we are initialing let's clear the record cache
     flag that is true iff we have the record in a cache */
     record_cache[i] = FALSE
     add 1 to num_recs
}
/* to retrieve the record */

target = <the desired time>
search time_index[] for TIMEi that is greater and return i-1
if record_cache[i-1] == FALSE
     calculate record offset position in file
          (header + recsize * (i-1))
     read record header into ephem_buffer_struct
     read data into ephem_data[]
     record_cache[i-1] = TRUE
endif
```

B - evaluate record at ephemeris time (return XYZ & X'Y'Z')

```
/*   now we evaluate the double precision difference array
     according to the black box by Fred Krogh @ JPL to return
     the STATE vector. Here state, is defined: double state[6]
     storing the position  & velocity.  The state is in the
     J2000 coordinate system */

state = magic_black_box()
```

3- get geocentric XYZ position of Earth Station

```
/*    Calculate the rotation matrix due to precession and
      nutation using the prcesj2() and nutate() with slight
      modifications the code. The code with modifications will
      be in the              function    rotation_matrix(),
      rot_pn will contain the new rotation matrix */

 If (rotation matrix is old)
 rot_pn[] = rotation_matrix()

/* s0[] contains the geocentric x,y,z position in J2000 */
/* v0[] contains the geocentric x,y,z velocity in J2000 */

s1[] = matrix_multiply(rot_pn[], s0[])
v1[] = matrix_multiply(rot_pn[], v0[])

/* s1[] will now contain the geocentric x,y,z position in local
      coordinates */
/* v1[] will now contain the geocentric x,y,z velocity in local
      coordinates */

/* now retrieve the local apparent sidereal time @ UTC
      midnight */
t_las = times.lst.midnight
modify t_las to be local sidereal time


/* now convert Satellite State to geographical coordinates */
/* Let s1[] = (X1, Y1, Z1) and Let sg[] = (Xg, Yg, Zg)     */
Xg = sg[0]; Yg = sg[1]; Zg = sg[2];
Xg = X1*cos(t_las) + Y1*sin(t_las)
Yg =-X1*sin(t_las) + Y1*cos(t_las)
Zg = Z1
sg[] = (Xg, Yg, Zg)

Xg = vg[0]; Yg = vg[1]; Zg = vg[2];
Xg = X1*cos(t_las) + Y1*sin(t_las)
Yg =-X1*sin(t_las) + Y1*cos(t_las)
Zg = Z1
vg[] = (Xg, Yg, Zg)

/* now lets calculate the vector from the Earth Station to the
    Satellite. */

es[] = eg[] - sg[]
vr[] = 0 - ( vg[] )
```

4- calc Azimuth & Elevation

We have the Earth Station X1,Y1,Z1 & the Satellite X2,Y2,Z2

get the vector V = (X1-X2, Y1-Y2, Z1-Z2)
convert vector V from rectangular to polar coordinates

5- add pointing corrections to Azimuth & Elevation
(from last peakup and station parameters)

```
azimuth = azimuth + azimuth_correct
elevation = elevation + elevation_correct
```