/N-08
55562
P. 142

NASA Technical Memorandum 110164

Manual for a Workstation-based Generic Flight Simulation Program (LaRCsim) Version 1.4

E. Bruce Jackson
Langley Research Center, Hampton, Virginia

(NASA-TM-110164) MANUAL FOR A WORKSTATION-BASED GENERIC FLIGHT SIMULATION PROGRAM (LaRCsim), VERSION 1.4 (NASA. Langley Research Center) 142 p

N95-30327

Unclas

G3 /08 0055562

May 1995

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Summary

LaRCsim is a set of ANSI C routines that implement a full set of equations of motion for a rigid-body aircraft in atmospheric and low-earth orbital flight, suitable for pilot-in-the-loop simulations on a workstation-class computer. All six rigid-body degrees of freedom are modeled. The modules provided include calculations of the typical aircraft rigid body simulation variables, earth geodesy, gravity and atmosphere models, and support several data recording options. Features/limitations of the current version include English units of measure, a 1962 atmosphere model in cubic spline function lookup form, ranging from sea level to 75,000 feet, rotating oblate spheroidal earth model, with aircraft C.G. coordinates in both geocentric and geodetic axes. Angular integrations are done using quaternion angular state variables. Vehicle X-Z symmetry is assumed.

A copy of this software is available upon request to the author.

Introduction

Historically, six degree of freedom aircraft simulations have been performed on larger minicomputers or mainframe computers due to limited processing speed and data storage capability on smaller workstation and desktop computers. With the advent of more powerful reduced instruction set computer (RISC) architecture, the processing capability of a desktop computer exceeds that of a supercomputer of a decade ago.

Simultaneously with the rise in popularity of workstation and desktop computers, the acceptance of UNIX-style operating systems has grown. This popular operating system has brought with it the C programming language in which the original UNIX kernal was written. While the standard C libraries lack some of the mathematical procedures of FORTRAN, in which most digital aircraft models are written, it is still possible to make use of this powerful and portable language. Abstract data types, longer variable names, data structures, and recursion allow the simulation architect to write maintainable and self-documenting software, with full access, through standardized library routines, to operating system capabilities in a nearly machine independent fashion.

Although not fully utilized in this version of LaRCsim, the popular X-Windows facility is easily manipulated in C. This provides for graphical operator/user interface capabilities on any X capable terminal or personal computer terminal emulator (called a *window server*).

This version of LaRCsim utilizes a `curses`-based terminal interface, which will support almost all types of computer terminals. X-windows support is planned for later versions of LaRCsim. Also supported is a Silicon Graphics GL workstation interface that includes out-the-window scenery and heads-up display symbology. The pilot controls are provided through a mouse or, optionally, an analog-to-digital interface (driver code for the analog-to-digital interface is not included since the software depends upon the choice of host processor and interface hardware.)

Output options include time history information in ASCII text tab-delimited, Dryden's GetData .ASC1, or Agile-Vu "`.flt`" format; a fourth option will write the time history data into a text file suitable for execution by one of several popular controls analysis software tools. Any global or static local variable can be recorded. The recording module uses debugger symbol to access static or global variables at a user-selected frequency. Specification of variables to be recorded can be made at run-time.

Overview

What is LaRCsim?

LaRCsim is a set of C routines that implement a full set of equations of motion for a rigid-body aircraft in atmospheric and low-earth orbital flight. It is intended to be used with additional, user-provided subroutines (either FORTRAN or C) that describe the aerodynamics, propulsion system, and other flight dynamic elements of a specific air vehicle. Once combined with the vehicle-specific routines, LaRCsim provides a desktop- and/or cockpit-based near-real-time simulation of the vehicle for engineering analysis and control law development.

The six rigid-body degrees of freedom are modeled. The modules provided include all of the kinematic relationships, most of the conventional output variables, geodesy and atmospheric models, and a data recording option. Some features/limitations of the current version are as follows:

- English units of measure.
- 1962 atmosphere model in cubic spline function lookup form, ranging from sea level to 75,000 ft. Included in the model are density, speed of sound, and sigma.
- Rotating oblate spheroidal earth model, with aircraft C.G. coordinates in both geocentric and geodetic axes.
- Vehicle X-Z symmetry is assumed.
- Quaternions are used in determining the angular orientation (although equivalent Euler angles are also calculated) to avoid the singularity at ± 90 degrees pitch angle.
- Gravitational harmonic effects due to the earth's oblateness are modeled.
- Modular design allows user to incorporate modified atmosphere, turbulence, and steady winds into the simulation.
- Rotating machinery effects are not modeled.

Origin and Purpose

LaRCsim was developed as part of an engineering flight simulation facility at NASA Langley Research Center that is used to debug aircraft flight control laws. This facility, known as Advanced Controls Evaluation Simulator (ACES), is used in the Dynamics and Control Branch (DCB) and currently consists of a dual RISC processer Silicon Graphics Onyx computer with RealityEngine-2 graphics driving an evaluation cockpit with throttles and a side stick hand controller.

The LaRCsim routines are used to provide appropriate aircraft dynamic responses to flight control commands. The flight control laws may be written in C or Fortran. The equations of motion are based upon work by McFarland in reference 1. The axis frames and sign conventions comply with the ANSI/AIAA recommended practice as outlined in reference 2; geodesy calculations use the relationships outlined in reference 3, as well as a custom geocentric to geodetic conversion developed by the author. The atmosphere model is derived from data found in references 4 and 5; other physical constants were obtained from references 6 and 7. LaRCsim itself is based upon FORTRAN routines originally developed by the author for the U. S. Naval Air Test Center (now the Naval Air Warfare Center) under a project known as CASTLE (see reference 8); these routines have ties back to the NASA Ames FORTRAN simulation routines known as BASIC, written by McFarland and others.

It is intended that LaRCsim applications be capable of running both with a cockpit and pilot in the loop as well as in terminal interactive and batch modes. This version includes both a generic display terminal and Silicon Graphics GL-based keyboard/mouse interfaces in addition to an external cockpit interface.

Changes from version 1.3

The ACES facility is still being developed, and LaRCsim continues to evolve. This release, version 1.4, differs from version 1.3 as follows:

- Six-degree of freedom trim capability has been added.

- The default settings file has been renamed, and is automatically updated at the end of a session so LaRCsim "remembers" settings from the previous session.
- Initial conditions may be specified at by a flag on the command line.
- Time step and initialization flags are now passed to model routines.

Additional information on these changes is available in the README file, provided in the software distribution. Please see this file for more information on what is required to adapt a version 1.3 simulation model to version 1.4. This report details the requirements to implement a new version 1.4 simulation model.

Input files

Default settings file. LaRCsim is fairly self-contained, and does not require any special supporting files to run. It does, however, utilize one file if it is present in the default directory: if present, a file named *.simname* (also called the default settings file) specifies what parameters are to be recorded during the simulation run, what parameters are to be used to trim the vehicle and what parameters are to be set to zero by the trim algorithm. The settings file may specify a default initial condition to which the model is initialized if no other initial condition file is specified on the command line. This file is automatically updated at the end of a LaRCsim session to record any changes in these settings. A sample settings file is shown in figure 1.

In the present version of LaRCsim, the default settings file contains four sections of information: previous simulation operation settings, a list of parameters to record, the default trim parameters, and the default initial conditions. These sections are independent and may appear in any order.

The first few lines of the default settings file demonstrates the use of a pound sign (#) as the first non-blank character to denote a comment line; comments can appear on any line (as long as the first non-blank character is a #). Blank lines are ignored.

The third line in the file is the first line that is used by LaRCsim: "`sim`" appears on a line by itself to indicated the beginning of a list of simulation options that were in force at the end of the last session. This line is followed by "`0010`" on the next line by itself; this flag line indicates which version of syntax is used (presently version 1.0) so that future version of LaRCsim will be able to recognize and use older input files. The contents of this section indicate what type of files to record at the end of the simulation session; the spacing with which to write the data files, the end time of the simulation; and the update rates for the model, screen refresh, and data recording; and how long (in seconds) the data buffer should be. In the example given in figure 1, a data file in matrix format will be written when the simulation ends. It will contain up to one hour's worth of simulation data, recorded at 20 Hz and every frame will be written to the data set. The model itself will run up to one hour, at 120 Hz, and the video screen (or terminal screen) will be updated at 30 frames per second.

In the next section, "`record`" appears on a line by itself to indicate the beginning of a list of parameters to be recorded during the simulation session. The next six lines are parameter declarations; these six parameters, if successfully located in the debugger symbol tables, will be added to 19 predefined variables and recorded during the simulation session.

The first three declaration lines are examples of how to specify scalar parameters. Note that these declarations are **local** variables to each routine. LaRCsim, by way of compiler-provided symbol tables, can locate and track the value of any local or global variable, but the variables must be **static** variables, declared as such at the top of each function. If the variables are **automatic** (i.e., not static), then the variable is defined only as long as the program is executing that function; thus, LaRCsim is unable to track automatic variables. The third declaration, of variable `forward_mu` in function **navion_gear**, is actually an automatic variable (in

```
# .navion created at 950406 22:57:12 by bjax
#============================ sim
sim
0010
    write_av        0
    write_mat       1
    write_tab       0
    write_ascl      0
    write_spacing   1
    end_time        3600.000000
    model_hz        120.000000
    term_update_hz  30.00000
    data_rate       20.000000
    buffer_time     3600.0000
end
#============================ record
record
0010
    aero        elevator
    aero        aileron
    gear        forward_mu
    *   generic_.f_gear_v[0]
    *   generic_.f_gear_v[1]
    *   generic_.f_gear_v[2]
end
#============================ trim
trim
0010
  controls: 3
#   module    parameter   min_val   max_val   pert_size
    *   generic_.euler_angles_v[1] -7.853981E-01   7.853981E-01    1.000000E-02
    aero        long_trim       -1.000000E+00   1.000000E+00    1.000000E-02
    *   cockpit_.throttle_pct   0.000000E+00    1.000000E+00    1.000000E-02
  outputs: 3
#   module    parameter   trim_criteria
    *   generic_.omega_dot_body_v[1]    5.000000E-05
    *   generic_.v_dot_body_v[0]        5.000000E-04
    *   generic_.v_dot_body_v[2]        5.000000E-04
end
#============================ init
init
0010
  continuous_states: 22
#   module    parameter   value
    *   generic_.geodetic_position_v[0] 2.374953E-04
    *   generic_.geodetic_position_v[1] 7.714288E-07
    *   generic_.geodetic_position_v[2] 1.099708E+01
    *   generic_.v_local_v[0]    1.740701E+02
    *   generic_.v_local_v[1]    1.522121E+03
    *   generic_.v_local_v[2]    -3.972784E+00
    *   generic_.euler_angles_v[0]      -1.481027E-04
    *   generic_.euler_angles_v[1]      1.127979E-01
    *   generic_.euler_angles_v[2]      2.089291E-03
    *   generic_.omega_body_v[0]        5.395570E-06
    *   generic_.omega_body_v[1]        0.000000E+00
    *   generic_.omega_body_v[2]        -2.788522E-05
    *   generic_.earth_position_angle   0.000000E+00
    *   generic_.mass    8.547270E+01
    *   generic_.i_xx    1.048000E+03
    *   generic_.i_yy    3.000000E+03
    *   generic_.i_zz    3.530000E+03
    *   generic_.i_xz    0.000000E+00
    *   generic_.d_cg_rp_body_v[0]     0.000000E+00
    *   generic_.d_cg_rp_body_v[1]     0.000000E+00
    *   generic_.d_cg_rp_body_v[2]     0.000000E+00
    aero        long_trim       -1.365538E-03
  discrete_states: 0
#   module    parameter   value
end
```

Figure 1. A sample default settings file.

the example simulation), and thus LaRCsim will complain when it reads this input file and attempts to locate `forward_mu` for the first time.

A local static variable is specified by the name of the function or subroutine in which it exists (e.g. `aero` or `navion_gear`) and the name of the variable. Case is important. `Elevator` is not the same variable as `elevator`.

The next three lines are examples of **global** variables; these are variables that have been declared outside the scope of a function. They are identified to LaRCsim as global by use of the `*` in place of a function name.

These last three lines also demonstrate the capability of LaRCsim to parse and locate elements of complex data structures; here, the elements of the landing gear force vector, `f_gear_v`, itself a part of the global data structure `generic_` will be added to the list of variables to record. The syntax for non-scalar data elements follows that of ANSI C. Arrays are all zero-index-based, as in C (unlike FORTRAN).

The `end` word must appear on a line by itself to delimit the list of recording variables that began with `record`.

The next section of the default settings file tells LaRCsim how to attempt to trim the vehicle when requested. The format is similar to that used by the `record` section, with the addition of a count of how many controls and how many output variables are specified (on the `controls: 3` and `outputs: 3` lines). Note: in this version of LaRCsim, the number of controls *must* equal the number of outputs. LaRCsim presently supports trim strategies with up to ten controls and outputs; in practice, however, no more than six are required for a rigid fixed-wing aircraft. See the section below for a description of the trim method and suggested techniques.

Each trim `control` specification includes a module and parameter name, as before for `record` specifications, as well as minimum and maximum values and perturbation size (see the Trimming Strategies section below for more information about these values).

Each trim `output` specification includes a module and parameter name and a criteria value that specifies how close to zero the output must driven by the trim algorithm before a successful trim is achieved.

The next section of the settings file, the `init` section, specifies what parameters are considered states, and should include both continuous states and discrete states (flags, Booleans, and integers), as well as a specification for the default values of these states. The initial condition described in this settings file do not have to describe a trimmed flight condition. Each line of the `init` section includes a module and parameter name, as before, as well as the initial value for that state.

Overriding the default settings. The user may specify on the command line, with the `-i` option flag, a different settings file with an alternate initial condition (IC) description. An IC settings file should have a file name that describes the initial condition, and end with a `.ic` file type, such as `on_ground.ic`, `two_mile_final.ic`, etc. The contents of this file are identical in format to the `init` section of the default settings file; LaRCsim will substitute the optional initial conditions for those found in the default settings file.

As an example, the command line

```
navion -i on_ground.ic
```

will cause the navion simulation to start at a specified initial condition defined in an IC settings file named `on_ground.ic`.

Similarly, the default trim strategy may be replaced with a new one by identifying a file containing the new `trim` portion of the settings file using the `-i` flag. By convention, the trim settings file should end in `.trim` and contain only a `trim` specifications section.

Additional parameters may be added to the list of recorded parameters by specifying (again with the `-i` flag) a file that contains a `record` specification. Any parameters thus specified will be added to the existing list of recorded parameters.

In the present version of LaRCsim, only one settings file may be specified at run time; it is possible to combine several settings file into a single file, and specify that file name at run time to achieve the desired set of trim parameters, recorded variables, and initial conditions.

Optional search path and redirection. At startup, LaRCsim will search the directories listed in an environment variable `LARCSIMPATH`, if it is defined, to find both the default settings file (e.g. `.navion`) and any specified settings file files (e.g. `on_ground.ic`). LaRCsim will use the first occurance of these files discovered in the path of directories specified by `LARCSIMPATH`. The variable `LARCSIMPATH` should be a colon-separated list of directories, similar to standard UNIX `PATH` environment variables. If `LARCSIM` is undefined, only the default directory will be searched to find the settings file.

A settings file may contain a line beginning with '`@`'; this indicates to LaRCsim an additional file that should be parsed. For example, the default settings file for the terminal version of a simulation (e.g. `.navion_term`) could contain the single line, `@.navion`; LaRCsim would interpret this to mean the contents of `.navion` should be parsed instead of `.navion_term`. (Note: `.navion_term` should be set to read-only to prevent it from being overwritten at the end of the LaRCsim session.)

The file pointed to by the indirection flag '`@`' could itself contain an additional indirection flag; caution should be used to avoid circular references.

Output files

.simname This default settings file, if it does not already exist, is created at the end of each simulation session and will contain the default values for record parameters, trim controls, and initial conditions. If the default settings file already exists and is not write-protected it will be replaced with a new copy.

`run.flt` This file, if requested with the `-a` flag, will be generated at the end of a session and will contain a time history of each recorded parameter in Agile-Vu format.

`run.m` This file, if requested with the `-r` flag, will be generated at the end of a session and will contain the time history information in matrix notation, suitable for use as a script in one of the popular control system design and analysis products.

`run.asc1` This file, if requested by use of the `-x` command line switch, wil be generated at the end of a session and will contain the time history information in a format understood by the Dryden Flight Research Center's `GetData` and `XPlot` tools.

`run.dat` This file, if requested with the `-t` command line switch, will contain ASCII tab-delimited columns of the recorded data; the first line contains the names of the parameters included. This format may be useful for importing time history data into spreadsheet or other charting programs.

Running a LaRCsim Example

Compiling LaRCsim

Building LaRCsim from the distribution is straightforward:

1. Define an environment variable, `LARCSIM`, to point to the source directory for the main LaRCsim routines. This should probably be done in the user's `.login` file (Example: `setenv LARCSIM /aces/larcsim/v014`)
2. Change the default directory to `$LARCSIM`.
3. Enter the command "`make`." This will:
   a. create a new object library file, `libls.a`
   b. compile all of the LaRCsim source files

c. put all the generated object files in the `libls.a` archive library

The object archive library `libls.a` only needs to be rebuilt after a LaRCsim module has been modified.

Compiling and building the example simulation

Once the `libls.a` file has been built in the `$LARCSIM` directory, move to the directory containing the aircraft files (in the case of the example simulation, move to the `navion` directory).

1. Enter the command "`make`" (for Silicon Graphics-based simulations) or "`make term`" for a terminal-based simulation. This will compile all the `navion` source files and link them together to form the executable simulation program `navion` (for Silicon Graphics-based simulations, or `navion_term`, for a terminal-based simulation).
2. If desired, create a default settings file in the format described above. It should be named *.simname*, where *simname* is the name of the executable simulation program.

Running the example simulation program

Typing `navion` on the IRIX command line will run the navion example simulation program on the GL console; the `navion_term` command will run the navion example simulation on most terminals.

Command line switches. The command for running a LaRCsim model may include a number of optional flags or switches:

- `-A` Run in conjunction with ACES cockpit (valid only for DCB users).
- `-k` Run on the Silicon Graphics console using the mouse as a joystick (`-k` and the `-A` flags are mutually exclusive).
- `-i` *filename*`.ic` Identifies an optional settings file that contains an alternate initial condition, trim strategy, or additional parameters to be recorded.
- `-f` *<iteration rate>* Specifies an iteration rate, in iterations per second, that the simulation model is to execute. Default frame rate is 120 iterations per second.
- `-o` *<output rate>* Specifies the rate at which the terminal or GL display screen should be updated, in frames per second. This rate must be an integral sub-multiple of the *iteration rate* (see `-f` above). For example, if the simulation model *iteration rate* is 120 iterations per second, legitimate choices for *output rate* are 120, 60, 40, 30, etc. frames per second (corresponding to $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}$, etc. of the *iteration rate*). Default screen refresh rate is 20 frames per second.
- `-e` *<end time>* Specifies an end time for the simulation run. The simulation will terminate when this value of simulated time is reached, if the simulation is not reset prior to that time.
- `-b` *<buffer length>* Specifies the length of the data storage buffer, in seconds. This circular buffer retains the last *buffer length* seconds of time history data. If not specified, the default *buffer length* equals the simulation *end time* given by `-e` above.
- `-s` *<storage rate>* Specifies the rate, in records per second, at which the requested parameters will be recorded to the circular data buffer. This rate must be an integral sub-multiple of the *iteration rate* (see `-f` above). For example, if the simulation model *iteration rate* is 120 iterations per second, legitimate choices for *storage rate* are 120, 60, 40, 30, etc. records per second (corresponding to $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}$, etc. of the *iteration rate*). If not specified, the default *storage rate* will be one-eighth of the *iteration rate* of the simulation model.

`-a` *<filename>* Specifies that an Agile-Vu compatible ".flt" file is to be written at the end of the session. Default filename is `run.flt`. If this option is the last one on the command line, a filename must be specified.

`-t` *<filename>* Specifies that a tab-delimited ASCII listing of time history data be written at the session. Default filename is `run.dat`. If this option is the last one on the command line, a filename must be specified.

`-x` *<filename>* Specifies that a GetData/X-Plot compatible ".ascl" file is to be written at the end of the session. Default filename is `run.ascl`. If this option is the last one on the command line, a filename must be specified.

`-r` *<filename>* Specifies that a matrix manipulation software compatible `.m` file is to be written at the end of the session. Default filename is `run.m`. If this option is the last one on the command line, a filename must be specified.

`-d` Specifies that the run allow interactive debugging; this prevents scheduling of timer interrupts and forces the GL display into single-buffer mode. This switch is probably not of great use to the typical user.

GL console operation. The command `navion -k` will bring up the out-the-window view, on the SGI console, with a heads-up display (HUD) overlay, and allow the user to maneuver the aircraft using the mouse and keyboard. The mouse movement simulates a control stick: push forward to move the stick forward, left to roll left, etc.

When the simulation first comes up, the aircraft is placed in the specified initial condition and the display will indicate the simulation is paused (on a GL display, this is indicated by the HUD symbology showing up in a red color). At this point the simulation may be trimmed (using the 't' key) or put into operation (with the 'p' key). A trim may be requested at any time during a run by use of the 't' key; this allows the vehicle to be flown to an interesting point of the sky and retrimmed. A successful trim will cause the current flight conditions to be remembered as the new initial condition.

At any point, the 'r' key will reset the simulation to the last remembered initial condition, allowing repeated landing attempts, for example.

The simulation may be paused at any point by use of the 'p' key to toggle between pause and run modes. Data is recorded in run mode and during trim attempts.

The simulation session will last for up to 60 minutes; a longer period of time may be specified on the command line as a parameter for the `-e` option (see the previous section for information on various command line options).

Pressing the escape key causes the simulation to terminate, and any recorded data will be written to the requested output files.

Display terminal operation. The command `navion_term` will operate the same simulation, but does not use a mouse or provide GL graphics. Instead, a simple instrument panel is presented on the user's terminal screen and several keyboard keys are pressed into service for flight controls. Figure 2 shows the screen used in LaRCsim version 1.4, with flight control keys indicated. No rudder command is available in this version.

External cockpit operation. The command `navion -A` will operate the same simulation, but LaRCsim will call the external cockpit interface routine to provide control stick, rudder pedal, and throttle positions, as well as pause and reset buttons. Most keyboard commands will still operate.

Note for DCB users: in the ACES cockpit, the upper red button on the handgrip resets the simulation, and the thumb button pauses the simulation.

```
L a R C S I M navion_term                                    0:00:00.0
Mach     0.007   Psi      0.1    NZ-G    0.997
KEAS       4.3   Thet     0.4    Alt         4    Alpha     0.42
Throt        0 % Phi      0.0    Hdot    0.000    Beta      0.03
Elevator       0.00   Aileron      0.00   Rudder        0.00

                                  stick

                                    i
                     throttle                     quit
                                    |
                      -a   +s    j -k- l         <ESC>
                                    |

                                    <
```

Figure 2. Terminal mode display

Trimming strategies

The trim algorithm, new to this version of LaRCsim, uses up to ten user-specified "controls" to drive a like number of "outputs" to values near zero. LaRCsim also forces pitch rate to zero prior to each trim attempt, so trimmed turns are not currently possible. Steady-heading sideslip trims, however, are possible and have been demonstrated. On-ground longitudinal trims are also supported.

The current mechanism to specify (and modify) the trim method requires editing the default settings file, or specifying a settings file containing a different set of trim controls and outputs by use of the `-i` flag on the command line. Listed below are examples of trim specifications that have been tested and used successfully in LaRCsim simulations at Langley Research Center.

In-flight longitudinal trim. In this example, pitch attitude, throttle, and a local variable in the aerodynamics module called "long_trim" are used to zero out the accelerations in pitch and body-X and -Z axes:

```
trim
0010
controls:  3
# module    parameter  min_val  max_val  pert_size
  *  generic_.euler_angles_v[1]  -0.785  0.785  1.0E-02
  aero  long_trim  -1.0000E+00  1.0000E+00  1.0000E-02
  *  cockpit_.throttle_pct  0.0000E+00  1.0000E+00  1.0000E-02
outputs:  3
#  module  parameter  trim_criteria
  *  generic_.omega_dot_body_v[1]  5.0000E-05
  *  generic_.v_dot_body_v[0]  5.0000E-04
  *  generic_.v_dot_body_v[2]  5.0000E-04
end
```

On-ground trim. With this strategy, two controls (pitch attitude and altitude) are used to obtain zero pitch and vertical acceleration, regardless of the aircraft's velocity or heading:

```
trim
0010
controls:  2
#  module  parameter  min_val  max_val  pert_size
  *  generic_.euler_angles_v[1]  -0.785  0.785  1.0E-02
```

```
  * generic_.geodetic_position_v[2]  0 30 0.0001
outputs:  2
# module  parameter  trim_criteria
  *  generic_.omega_dot_body_v[1]  5.0000E-05
  *  generic_.v_dot_local_v[2]  5.0000E-04
end
```

Steady-heading sideslip trim. In this strategy, three pilot control trim variables are used, along with throttle, pitch attitude, and heading angle to achieve zero accelerations in angular and local velocities:

```
# this trim is for steady-heading sideslip, where
# sideslip is given by local velocities.
trim
0010
controls:  6
#  module  parameter  min_val  max_val  pert_size
  subsystems longtrim  -3.0000E+01  3.0000E+01  3.0000E-02
  *  generic_.euler_angles_v[1]  -0.5  0.5  1.0000E-03
  *  cockpit_.throttle_pct  0.0000E+00  1.0000E+00  1.0000E-03
  subsystems lattrim  -10 10 0.01
  subsystems pedtrim  -10 10 0.01
  *  generic_.euler_angles_v[0]  -0.5  0.5 0.001
outputs:  6
#  module  parameter  trim_criteria
  *  generic_.omega_dot_body_v[0]  5.0000E-05
  *  generic_.omega_dot_body_v[1]  5.0000E-05
  *  generic_.omega_dot_body_v[2]  5.0000E-05
  *  generic_.v_dot_local_v[0]  5.0000E-04
  *  generic_.v_dot_local_v[1]  5.0000E-04
  *  generic_.v_dot_local_v[2]  5.0000E-04
end
```

Creating a New Aircraft Simulation

Mandatory routines

A new simulation model must provide, as a minimum, an aerodynamics routine with an entry point labeled `aero()`. The source code is usually kept in a file named after the specific vehicle, e.g. `navion_aero.c`. In addition, a complete vehicle model would include `engine()`, `subsystems()`, `inertias()`, and `gear()` routines, although stub routines are provided for these.

Inputs to these routines come from the `GENERIC` global variable structure, for which useful aliases are provided in the `ls_generic.h` header file (see Appendix A). The more sophisticated models will undoubtedly create an aircraft-specific set of global variables; the use of a `struct` or `COMMON` is recommended to share these global specific variables between simulation components. Interface to the simple keyboard, mouse and/or ACES cockpits is available through the `COCKPIT` data structure.

The expected outputs from `aero()` are simply the aerodynamic forces and moments about the reference point, in lbs and ft-lbs, respectively, being stored in the `F_aero_v` and `M_aero_v` vectors (scalar names `F_X_aero`, `F_Y_aero`, `F_Z_aero`, `M_l_aero`, `M_m_aero`, and `M_n_aero`).

Likewise, the outputs from any `engine()` or `gear()` routines should be stored in the `F_engine_v`, `M_engine_v`, `F_gear_v`, and `M_engine_v` vectors as appropriate. Refer to the example simulation for samples of how to do this.

If desired, the LaRCsim user may craft an `inertias()` routine to keep track of fuel burn (using an aircraft specific fuel flow parameter provided from `engine()`)

and adjust the inertia properties and center of gravity location values kept in GENERIC: Mass, I_xx, I_yy, I_zz, I_xz, and vector quantity D_cg_rp_body_v (the location of the center of gravity, measured from the reference point, in body axis); for most simulation studies of an engineering nature, the fuel quantity is a constant that can be, along with mass properties and C.G. location, be set at initialization (through user routine model_init(), or through a settings file.).

The user *must* have a model_init() routine, which is called before each simulation run, to set certain parameters. See the section below for a list of necessary parameters. Failure to set certain parameters will lead to an immediate divide by zero error, or unreasonable dynamic response of the simulation.

The subsystems() hook allows control system models, navigation system models, sensor models, autopilots, etc. to be included in the more elaborate simulations. These routines will likely use some of the parameters provided in GENERIC and get other inputs from and store outputs to user-defined common memory structure(s).

Mandatory parameters

The following is a list of the variables for which the user-supplied vehicle routines *must* provide reasonable values:

Mass	vehicle inertial properties;
I_xx	these must be non-zero
I_yy	
I_zz	
I_xz	
D_pilot_rp_body_v	pilot location w.r.t. reference point
D_cg_rp_body_v	C. of Grav. location w.r.t. reference point
F_aero_v	aero forces, body axes
F_engine_v	engine forces, body axes
F_gear_v	gear forces, body axes
M_aero_v	aero moments, body axes, about ref. pt.
M_engine_v	engine moments, body axes, about ref. pt.
M_gear_v	gear moments, body axes, about ref. pt.
Runway_altitude	location of threshold of runway of interest
Runway_latitude	
Runway_longitude	
Runway_heading	

These values may be initialized once, in the model_init() function, or may be calculated each frame, in a procedure called by ls_model(). The mass properties must by non-zero to avoid mathematical errors.

The following variables should be specified in model_init() to the appropriate initial conditions; they are thereafter calculated by the EOM routines:

Geodetic_position_v	geodetic position in radiansfeet
Euler_angles_v	aircraft attitude (ϕ, θ, ψ), radians
V_local_v	center of gravity velocities, in ft/s
Omega_body_v	body axis rates, in rad/s

where geodetic position is latitude, longitude, and altitude above sea level. The following variables may be set by the user routines if desired:

V_local_airmass_v	airmass velocity: steady wind
V_local_gust_v	body axis turbulence

Support for FORTRAN routines

Existing FORTRAN routines can be interfaced to LaRCsim through use of "wrapper" routines that translate between existing FORTRAN `COMMON` data structures and the `GENERIC` and other LaRCsim data structures. It is possible to write FORTRAN versions of `aero()`, `engine()`, `inertias()`, etc., but the reader is encouraged to write new models in C (or even C++) for maintainability and compatibility reasons.

The secret to writing these "wrapper" routines is to realize that, at least in IRIX, FORTRAN entry points and commons appear (from the C side) as having the same name that they do in FORTRAN, but in lowercase and with an underscore ('_') appended, and vice-versa. Thus, a FORTRAN `COMMON` structure named `SIMPAR` will appear to the C language routine as a global variable named `simpar_` (it must be declared as an external global structure in the C routine or header file). Likewise, a FORTRAN subroutine declared as `SUBROUTINE PLSURF` can be called from a C program as `plsurf_()`. Consult the documentation for each particular operating system for more information on how to develop a "wrapper" for an implementation on that system.

When the real-time loop is entered, the routines specified in `ls_model()` are called once per loop. The user is expected to replace the simple `aero()` and `engine()` routines provided in this package with more realistic aerodynamic and propulsion system models. These models should calculate, based upon the current Mach, altitude, angle of attack, etc. the appropriate forces and moments due to aerodynamics, engines, and perhaps landing gear, if appropriate. These forces and moments are to be provided in units of lbs and ft-lbs, in the X-Y-Z body axis system (positive indicates forward, right, and down, respectively) acting at the predefined reference position. If fuel consumption or weapon drops are to be simulated, an `inertias()` routine should be added, and the values of `Mass`, `I_xx`, `I_yy`, `I_zz` and `I_xz` should be updated in each loop. Center of gravity movement should be reflected in updates to the `D_cg_rp_body_v` vector as well. It is also possible to change runway location during simulation operation, if appropriate; the code to provide this capability is not included in the present LaRCsim version, however.

Function Data Interpolation

Overview. Mathematical descriptions of the aerodynamics of most flight vehicles usually include non-linear elements, such as the stall "break" characteristic exhibited by straight fixed-wing aircraft at higher angles of attack. Other aerodynamic properties exhibit even more pronounced non-linearities with respect to angles of attack, sideslip, Mach number, control surface deflection and other "independent" flight conditions. Other components of a flight vehicle model, such as propulsion systems and control law gain tables, often need to represent a very non-linear parameter in some fashion.

Many ways have been developed in previous years to represent these non-linear functions, including specialized mechanical analogues and electrical circuits. In present flight simulators these functions are represented through special-purpose software. To save memory, early software-based functions were generated using polynomials to approximate the non-linear characteristics of the actual airplane. As memory became less expensive, small tables of numbers were stored and then interpolated at run time. The present industry practice is to use large amounts of memory to store multi-dimensional tables; a return to polynomial representation may be underway to generate models that are mathematically smooth (see reference 10). The atmosphere model developed for LaRCsim uses a combination of these techniques; it represents atmospheric properties by use of a table, based upon altitude, of the coefficients of a set of cubic spline functions that provide smoothly varying curves that agree with the original atmosphere model at the "knots".

To provide a general, C-based function generation capability, the `ls_funcgen.c` module was developed. This simple code makes use of an object paradigm to represent the function tables and a recursive C-routine to perform the interpolation along each dimension. This particular solution is, in the opinion of the author, elegant in its object-oriented design, recursiveness and the capability to handle function sets of unlimited size and dimension; it is, on the other hand, a little difficult to understand, and not as fast as an in-line, non-recursive, FORTRAN routine used for comparision.

To become really useful, a set of tools to generate the function data code for a particular simulation would be nice and may become available in a later version of LaRCsim.

Terminology. The following terms are used to describe the function generation routine:

Breakpoint data set A monotonically increasing vector of real numbers that represent the values of an independent variable for which the dependent function is known and tabulated.

Dependent variable The value of the function, or the return value from the function generation subroutine. Known values of the dependent variable for specific values of the independent variable(s) upon which it depends are provided by the user in the form of data tables; the routines described in this section provide linearly interpolated values of the dependent variable for an arbitrary set of independent variable values.

Dimension Each dimension of a data table represents an independent variable upon which the dependent variable, represented as points in the function table, are based.

Function table A multi-dimensional table of dependent variable values that correspond to a given number of breakpoint data sets. In LaRCsim, the first dimension varies most rapidly.

Independent variable An argument to the function. In terms of aerodynamic tables, the independent variables are usually one or more of the following: angle of attack, angle of sideslip, Mach number, and control surface deflection.

Index and weights value A floating point number, corresponding to a specific breakpoint set, that represents the present location of the independent variable in that breakpoint set. The integer before the decimal represents the index (0-origin based) of the breakpoint data point that is closest to, but less than, the actual independent variable value; the fractional portion of the number represents the fractional distance the independent variable is between the indexed and next-higher breakpoint value. It is defined as w, where

$$w = i + d$$

where d is the interpolation ratio given below and i is the current index of the next-lower value of the breakpoint set.

Interpolation ratio This fractional quantity, d, represents the location of the independent variable between the next lower and next-higher values of the breakpoint set. It is defined as:

$$d = \frac{x - x_i}{x_{i+1} - x_i}$$

where x is the value of the independent variable, x_{i+1} is the next-higher value of the breakpoint set, and x_i is the next-lower value of the breakpoint set.

Normalization The process of determining the proper index and weights value w (see above) for the present independent variable value.

Implementation. If one were to describe the problem of data interpolation, one might use the following description:

> The value of a function is represented in an orthagonal N-dimensional table. Each dimension of the table corresponds to a monotonicly increasing independent breakpoint variable. The data in the table is arranged such that each entry represents the known value of the function, or dependent variable, corresponding to fixed value(s) of the breakpoint, or independent variable(s), at that index of the table. The problem is to determine the value of the dependent variable at any arbitrary value(s) of the independent variable(s). This is done by interpolating the known value of the function between the two surrounding table entries; in effect, generating a new table entry. If multidimensional, this process may be repeated for each dimension of the table, but the "known" values used for each succeeding interpolation are actually interpolated values from the previous dimension. This recursion continues until the value of the dependent variable has been interpolated for the last dimension; this quantity is the value of the function corresponding to the arbitrary values of the independent variables.

In the most general case, some breakpoint sets may be shared between function tables; and since breakpoint normalization is relatively CPU intensive, re-use of normalized breakpoints is a good idea. Similarly, often times the function table itself may be duplicted to represent similar but independent functions; a common example is a set of spoilers on an aircraft that are operated independently, where the spoilers have similar or identical aerodynamic effect (except for perhaps a minus sign) but may well be operated at different deflections.

The function generator data structures used in LaRCsim allow for re-use of breakpoints and function table data; for this reason, understanding the data structures may take a little examination and thought. Separate "objects" that represent the breakpoint sets, the function values themselves, the actual function data (which associates the function data with the corresponding breakpoint sets) and the final object, the non-linear function (which associates function data with breakpoint normalization data) are all stored as separate data structures, as described below.

In keeping with the object-oriented abstraction of the problem, breakpoint data sets and function tables are stored separately in `BREAKPOINTS` and `DATA` structures. They are associated together in an individual `FUNC_DATA` structure; the `FUNC_DATA` structure is an abstraction of a multi-dimensional curve or surface. These data structures are defined in the header file `ls_funcgen.h`.

The `NONLINEAR_FUNCTION` structure associates this function data with the interpolation information (index and weights as well as the last value returned on the previous lookup call). This structure is an abstraction of the process of interpolating a `FUNC_DATA` curve; it includes a pointer to the function data as well as state information about where the function was most recently found, which speeds up subsequent searches since a sequential search through the breakpoint vector, starting with the last index used, is used instead of a binary search. The crawl search is believed to be better for flight simulation function generation applications than a binary search, since the traditional independent arguments change fairly slowly.

The tables are effectively unlimited in size and number of dimensions; the maximum length in any dimension is set by `MAX_LENGTH`, and the number of dimensions is set by `MAX_DIMENSION`; both are declared in the `ls_funcgen.h` header file.

Another data structure, `ARG_LIST`, is used to pass interpolation information to the lookup function. It contains the current index value and interpolation ratio for each dimension of the nonlinear function.

For an example implementation of these data objects and an actual implementation of this code, refer to the header information found in `ls_funcgen.c`.

Implementation Details

File Descriptions

The source and header files that make up the LaRCsim application are listed below, along with individual file version numbers:

In the LARCSIM directory:

```
Makefile, v 1.0
ls_ACES.h, v 1.4
ls_cockpit.h, v 1.3
ls_constants.h, v 1.0
ls_err.h, v 1.1
ls_funcgen.h, v 1.1
ls_generic.h, v 1.0
ls_matrix.h, v 1.1
ls_sim_control.h, v 1.11
ls_sym.h, v 1.9
ls_tape.h, v 1.6
ls_types.h, v 1.0
LaRCsim.c, v 1.4.1.7
atmos_62.c, v 1.0
default_model_routines.c, v 1.3
ls_ACES.c, v 1.8
ls_accel.c, v 1.5
ls_aux.c, v 1.12
ls_err.c, v 1.2
ls_funcgen.c, v 1.6
ls_geodesy.c, v 1.5
ls_gravity.c, v 1.2
ls_ifgl.c, v 1.15
ls_ifterm.c, v 1.1
ls_init.c, v 1.4
ls_matrix.c, v 1.1
ls_model.c, v 1.3
ls_record.c, v 1.11
ls_settings.c, v 1.6
ls_step.c, v 1.5
ls_sym.c, v 2.7
ls_sync.c, v 1.7
ls_trim.c, v 1.9
ls_writeasc1.c, v 1.7
ls_writeav.c, v 1.10
ls_writemat.c, v 1.11
ls_writetab.c, v 1.4
```

In the example directory:

```
Makefile, v 1.0
navion.h, v 1.3
.navion, v 1.0
navion_aero.c, v 1.0
navion_engine.c, v 1.1
navion_gear.c, v 1.0
navion_init.c, v 1.0
```

Each of these components of the LaRCsim simulation program are described below.

Compilation support files.

`Makefile` A simple makefile that allows the LaRCsim object library `libls.a` to be created and/or updated on most Unix platforms by the simple command `make`. To build the example simulation, issue the **make** command in the LaRCsim directory, and then move to the navion subdirectory and issue another **make** command.

Header files.

ls_ACES.h This header file describes various constants and data structures used with the Dynamics and Control Branch Advanced Controls Evaluation Simulator (ACES) hardware; it is not of interest to a non-DCB user.

ls_types.h This file defines the two principal data types used in LaRCsim: **SCALAR** and **VECTOR_3**. The **SCALAR** data type, which is defined as a **double**, is suggested for use by any C or C++ routines added to LaRCsim. This definition allows easy modification of the level of precision of calculations, since changing the type definition of **SCALAR** in this routine to, say, **float**, would halve the precision of all LaRCsim module calculations.
Prior to version 1.3, the scalar floating-point type **DATA** was defined, but is not recommended for further use to avoid confusion with the FORTRAN compiler directive of the same name. It remains defined in this module for commonality with older routines, but may be removed in future versions.
A 3-element vector of **SCALAR** elements, **VECTOR_3**, is defined for use by routines which may benefit from using vector notation. Many of the components of the **generic_** global variable structure are defined in terms of **VECTOR_3** elements, with an alternative set of three scalar names defined for convenience.

ls_constants.h This header file defines useful constants, such as **PI**, equatorial radius of the earth **EQUATORIAL_RADIUS** as well as its square **RESQ**, earth geodesy parameters **FP**, **E**, and **EPS**, the inverse of nominal gravitational acceleration **INVG**, the rotation rate of the earth, **OMEGA_EARTH** (in radians per second), useful conversion factors **V_TO_KNOTS**, **DEG_TO_RAD**, and **RAD_TO_DEG**, and standard sea-level atmospheric density, **SEA_LEVEL_DENSITY**, in English units (slug/ft^3).

ls_generic.h This header file defines the **generic_** aircraft parameter global structure which is used to pass global parameters between aircraft subsystem models and the various equations of motion routines. The generic parameters provide the common aircraft state information (positions and velocities) as well as other parameters such as accelerations, forces and moments, vehicle geometry, mass and inertia, and atmospheric properties. A complete description of the contents of the **generic_** data structure is given in Appendix A.

ls_sim_control.h This header file defines the **SIM_CONTROL** global structure which is used to indicate command-line and other options set by the user. It contains the mode flag **sim_type** to indicate what mode of operation has been requested (**batch**, **terminal**, **GLmouse**, or **cockpit**), as well as information about run number, date and time stamps, and output formats requested for trajectory information.

ls_cockpit.h This header file defines the **COCKPIT** global structure which is used to pass pilot control position information between the cockpit (either a keyboard, mouse, or actual cockpit) and the rest of the simulation routines. Some abbreviations for locations within the **COCKPIT** structure are also provided for convenience.

ls_err.h This header file defines the **ERROR** global structure which is used to signal error conditions to the rest of the simulation. At present, the only errors defined are those relating to the table lookup routines defined in `ls_funcgen.c`.

`ls_funcgen.h` This header file provides prototypes for the linear interpolation (data table lookup) routines available in this version of LaRCsim. See the section "Function Data Interpolation" above for more information.

`ls_matrix.h` This header file provides function prototypes for general real matrix manipulation routines; it is used by the **`ls_trim`** routines.

`ls_sym.h` This header file provides prototypes for various symbol table lookup and manipulation routines, **`ls_findsym()`**, **`ls_put_sym_val()`**, and **`ls_get_sym_val()`**. This particular header file is probably of not much interest to the casual LaRCsim user.

`ls_tape.h` This header file defines the time-history data recording structure, **`tape_`**, which is used in the **`ls_record()`** and **`ls_write`**_xxx_**`()`** routines, and is of not much interest to the casual LaRCsim user. However, the number of parameters that may be stored is determined by the definition of **`MAX_TAPE_CHANNELS`** which is contained in this header file (currently set to 1024 parameters).

Routines called in the main execution loop.

`ls_accel.c` The first of three main EOM routines. This function sums the body-axis forces and moments provided by the **`aero()`**, **`engine()`**, and **`gear()`** routines (these are written by the user; example **`aero()`** and **`engine()`** routines are found in the file **`navion.c`** included in this package) and calculates the resulting total angular and linear accelerations in geocentric coordinates. Forces and moments are taken to act at the reference point, which is fixed to the body. The center of gravity location is defined relative to the reference point by variables `D[xyz]_cg` (found in vector **`D_cg_rp_body_v`**). The total angular and linear accelerations are corrected to act through and about the center of gravity.

`ls_step.c` This is the second of the three main EOM routines. This function performs the integration of the vehicle accelerations and velocities to form the updated vehicle velocities and positions. The time variable, **`Simtime`**, is integrated as well. The integration of accelerations uses a predictive (forward) integration; the integration of velocities is a modified trapezoidal backwards integration algorithm. These integration routines have been used successfully at NASA-Ames, NASA-Langley, and NATC/NAWC Patuxent River for many years and are well proven.

`ls_aux.c` This is the third major EOM routine. This function calculates most of the auxiliary variables based upon the updated vehicle state, including conventional accelerometer readings at both the C.G. and the pilot station, new values for angles of attack, sideslip, flight path, Mach number, gravity, and numerous descriptions of velocity and position in several axes. The state variables for geocentric latitude, longitude, and radius are converted to more useful geodetic (map coordinates) latitude, longitude and altitude (M.S.L.) as well as runway relative coordinates from a prespecified runway.

The next three routines are called by the main EOM routines to perform supporting calculations.

`atmos_62.c` The 1962 Standard Atmosphere Tables for density and speed of sound, in cubic spline lookup format, along with the necessary interpolation routines. Data is included from sea level to 240,000 ft.; however, the ambient temperature and pressure are described as parametric equations and are only valid to about 75,000 ft. in this version of LaRCsim.

ls_geodesy.c This function converts geocentric latitude and radius to geodetic latitude and altitude above sea level, and vice versa. It is based upon relationships provided in reference 3, which define the transformation from geodetic to geocentric; unfortunately, reference 3 doesn't include the opposite transformation, which is fairly complex. Since LaRCsim uses geocentric coordinates as *the* inertial axes set, and performs the translational integrations in the geocentric frame, it is necessary to have a means to efficiently convert back to geodetic coordinates, since these are the coordinates most often used for navigation (map latitude, longitude, and altitude). The `ls_geoc_to_geod()` routine, found in the `ls_geodesy.c` module performs this approximate conversion. Note: recently an engineering note in the AIAA Journal of Guidance, Control and Dynamics describes a closed-form solution; it is quite complex and has not yet been evaluated for this application (reference 9).

ls_gravity.c This routine calculates the value for local gravity, based upon geocentric latitude and radius, including effects due to oblateness of the earth (harmonics), based on equations given in reference 3.

The user-supplied aircraft model is called by the next routine.

ls_model.c This routine is an executive to the vehicle (user supplied) routines `engine()`, `subsystems()`, `aero()`, and `gear()`, or whatever set of routines the user decides are needed to adequately model the vehicle properties.

Any functions that are not satisfied by user-provided routines are provided by the next routine:

default_model_routines.c This module contains stub routines for what are normally user-provided functions, `inertias()`, `subsystems()`, `engine()`, and `gear()`. If these are not provided by the user, these stub routines satisfy the loader at link time, with no ill effects aside from fixed weight, thrust, and the lack of ability to land. The user *must* provide initial values of certain mass properties, as well as force and moment vectors, in a routine named `model_init()`. See the section above on creating a new model for more details on what parameters must be initialized by user software.

Data logging is provided by a call to the next routine:

ls_record.c This routine stores preselected global variables into a data structure for later playback or analysis. `ls_record()` automatically saves 19 channels of data (e.g. these outputs are hardwired) that contain state and basic input information from each run; in addition, the user can specify (through the settings file) additional parameters to record. Variables are addressed via memory locations found in the debugger symbol table of the executable; for this reason, the various modules that comprise a LaRCsim executable **must** be compiled and linked using the symbol table option (usually a -g switch). A LaRCsim simulation that can't locate a specified variable will complain at invocation, but continue to execute; those parameters that are not found will not be recorded. The data structure `TAPE` utilizes a circular buffer that, when full, begins to replace the oldest time history data with newer data. In the version 1.4 distribution of LaRCsim, this buffer records every eighth time slice.

Finally, interfaces with the pilot and synchronization with the real world are accomplished by the following routines:

ls_sync This module contains routines involved with synchronizing the operation of LaRCsim to match simulated time with real-world time on some UNIX platforms. The portability of this module is in question, however. It makes use of system services `signal()`, `setitimer()`, `pause()`, and the `itimerval` data structure, which are supported on both SGI (IRIX 5.2) and Sun (SunOS 4.1.3) platforms.

ls_ifgl.c This module contains an IRIS GL (Graphics Library) interface for interactive runs on Silicon Graphics computers (running IRIX 5.x), as well as dummy synchronization routines (which aren't needed if run under GL, since the drawing calls effectively synchronize to real-time). This module replaces `ls_ifsun.c` for Silicon Graphics implementations.

ls_ifterm.c This module contains a simple interface for interactive runs on most Unix computers, using the `curses` library of terminal routines, as well as routines to synchronize simulation with real-time, using standard unix system routines `setitimer()`, `signal()`, and `pause()`. It was the intent of the author to keep the routines very generic, without relying on either BSD or System V style system calls; our ignorance of these nuances may well show through, however; this routine works well on a SunSPARCstation-1 and -2, and will work on an SGI IRIS 4D machine.

ls_ACES.c This module contains driver code to communicate with the Advanced Controls Evaluation Simulator (ACES) cockpit used in the Dynamics and Control Branch, and is of little interest to the non-DCB LaRCsim user.

Support routines. The following routines provide additional services for the LaRCsim application, and are not typically called during the main simulation loop:

LaRCsim.c This routine is provided as an example executive function to call the appropriate routines in the proper sequence both prior, during, and at the end of a simulated run. `LaRCsim.c` includes the `main()` procedure for the simulation. It also interprets any command line options provided by the user, and initializes some simulation data structures with default values. At the conclusion of the simulation, it calls the output routines `ls_writemat`, `ls_writeav`, `ls_writetab`, and `ls_writeascl`.

ls_err.c This module reports errors in a semi-meaningful way. By properly loading the `ERROR` structure elements (see `ls_err.h`) and then calling `print_error()`, a LaRCsim routine can have an error message printed on stderr.

ls_funcgen.c The `ls_funcgen` module provides a simple linear interpolation routine for doing function generation using data tables. At present, this routine is limited to functions of six dimensions and 63 breakpoints along each dimension. It reports errors via the `ls_err()` routine. See the section above on "Function Data Interpolation" for more information on using this capability.

ls_init.c This routine calls the EOM functions and the user-supplied vehicle initialization routines in the proper sequence to initialize the vehicle prior to a run, or to reset at the end of a run.

ls_matrix.c This module contains several utilities to create, delete, print, and invert general real matrices. It is used by the trim routine.

ls_settings.c This module contains the code that deals with settings files. Two main routines are defined: `ls_get_settings()` and `ls_put_settings()`. A single parameter, `desired_file_name`, is accepted by `ls_get_settings()`. Calling `ls_get_settings()` with a file name specified will cause a search for a file by that name along the `LARCSIMPATH` directory path; if a null string is passed to `ls_get_settings()`, a default settings file with the name of the executable simulation program, prepended with a '.', will be hunted for along the path. If either file is found, that file will be opened, read into memory, and parsed by the `ls_parse_settings()` routine. A table of facilities is kept that provide entry points for both reading and writing each type of information (e.g. trim, init, record). `ls_parse_settings()` will call the appropriate routine as the designated keyword is found, passing a pointer to the appropriate location in the file buffer to that routine. If `ls_parse_settings()` encounters a line in which the first non-blank characters is '`@`', it will use the characters following the '`@`' sign as a file name, search for and open that file, and recursively call itself. A call to `ls_put_settings()` will create a default settings file, replacing the previous one, if it exists, and then calls each facilities' `put_settings()` routines, as kept by the facility table, in sequence, causing the current LaRCsim settings to be recorded.

ls_sym.c This routine performs symbol table lookups to resolve static local and global variable names into virtual memory addresses. It is used by `ls_record()` to record time history data during run time. It is not intended for use by the general LaRCsim user; and its portability is in question, as this capability is usually highly platform-dependent. It does appear to work on SGI (IRIX 5.2) and Sun (SunOS 4.1.3) operating systems, however.

ls_trim.c This module contains a Newton-Raphson algorithm for solving simultaneous non-linear equations. Given n "control" parameters, `ls_trim()` will perturb those parameters and observe the effect upon n other "output" variables. After measuring these partial derivatives, using a single-sided difference approach, the algorithm makes a constrained step of all n controls simultaneously to try to reduce the root-mean-square value of the sum of the n outputs. This process repeats for up to `Max_Cycles` or until all outputs are within a specified tolerance of zero.

ls_writeav.c This module writes time history data from the `Tape` data storage structure to a file named `run.flt` at the end of the simulation session. This data file is in a format recognizable to the Agile-Vu trajectory visualization tool developed for Silicon Graphics workstations by McDonnell-Douglas and the Naval Air Development Center. The `-a` command line switch will choose this output format; by default, no `run.flt` file is created.

ls_writeasc1.c This module writes time history data from the `Tape` data storage structure to a file named `run.asc1` at the end of the simulation session. This data file is in a format recognizable to the GetData and XPlot programs, written for X-windows machines by the kind folk at NASA Dryden Flight Research Center. (see reference 11 for information on this time history format.) The `-x` command line switch will choose this output format; by default, no `run.asc1` file is created.

ls_writetab.c This module writes time history data from the **Tape** data storage structure to a file named **run.dat** at the end of the simulation session. This data file contains a ASCII based, tab-delimited listing of each parameter at each recording point; these files can therefore become quite large for a long simulation session. The **-t** command line switch will choose this output format; by default, no **run.dat** file is created.

ls_writemat.c This module writes time history data from the **Tape** data storage structure to a file named **run.m** at the end of the simulation session. This data file is in a format recognizable to a typical commercial matrix manipulation application. The **-r** command line switch will choose this output format; by default, no **run.m** file is created.

The following routines, contained in a separate directory, provide an example aircraft simulation including simple aerodynamic, engine, and initialization routines.

navion.h This header file defines a data structure that contains the linear aero coefficients, **COEFFS**, which can be made available for run-time modification of the example aircraft's aerodynamic properties and stability characteristics.

navion_aero.c A simple, linear aerodynamics model of the North American Navion for a trimmed level flight at 100 knots.

navion_engine.c This file contains a simple **engine()** routine with an optimistic thrust calculation that allows the venerable Navion to break Mach 1 in level flight.

navion_gear.c This module includes a fairly simple landing gear (mass-spring-damper) model of tricycle arrangement, and is not representative of the North American Aviation Navion.

navion_init.c This module initializes the mass properties and sets forces and moments and velocities to zero. It also initializes elements of the pilot and cg displacement vectors (relative to the reference point).

Makefile This makefile is used to build either a GL-based (for Silicon Graphics machines) or terminal-based version of the navion example LaRCsim executable. Invoke with **make** to generate the GL-based executable (which will be named **navion**), or specify **make terminal** to create the **curses**-based executable, **navion_term**.

.navion This ASCII data file contains a list of any parameters that are to be added to the recorded parameters list, as well as the desired set of trim parameters and initial condition states and controls. This file shows an example of the format to be used, and may be opened and modified with a text editor.

Theory of Operation

Inspection of the LaRCsim code (see Appendix B), beginning with the **main()** routine found in module **LaRCsim.c**, will demonstrate how and in what order the software is called. The **main()** routine initializes the contents of the **sim_control_** data structure and certain execution variables, such as the local variables **endtime**, **speedup**, **io_dt** (the terminal refresh period), **multiloop** (the number of model loops per terminal refresh), and **model_dt** (the model iteration time step). A call is then made to **ls_get_settings()** which opens the default settings file, if it exists, allows it to override these hardwired default values.

ls_get_settings() parses the default settings file and makes calls to **ls_record_get_settings()**, **ls_trim_get_settings()**, and **ls_init_get_settings()**, each of which initialize their various data structures and parse the appropriate section of the default settings file. **ls_get_settings()** then returns control back to **main()**.

`main()` then makes a call a call to `ls_check_opts()` which looks at any command line arguments, allowing them to override the default settings, if appropriate. (If the `-i` flag is encountered, for example, another call is made to `ls_get_settings()`, this time passing the name of the requested optional settings file). `ls_stamp()` is then called to generate a time and date stamp for the simulation run. These are stored in the `sim_control_` data structure.

The `main()` routine then calls `ls_init()`, which sets `Simtime = 0` and then initializes the initial conditions data structure. If no initial conditions were specified in the default settings or optional settings file, the initial conditions data structure is set to contain information about the thirteen rigid body and environment states. `ls_init()` then uses the values of the initial conditions data structure to set the simulation to the specified initial condition and then calls `model_init()`, normally a user-supplied routine. The sample routine provided in this package is found in file `navion_init.c`; it initializes control positions, inertia properties, vehicle forces and moments, and vehicle positions and velocities. Routine `ls_init()` then calls `ls_step()` with a time step of 0 and the initialization flag set.

Responding to the initialization flag, `ls_step()` initializes the integrator internal states ("past values") to zero, converts the initial geodetic latitude, longitude, and altitude values into geocentric latitude, longitude and radius (from the center of the earth) values; corrects the eastward velocity component to account for earth rotation; initializes the quaternion variables based upon the present Euler angles; initializes the local-to-body transformation matrix; calculates local gravity; and calls `ls_aux()` so that the miscellaneous output variables (such as angles of attack and sideslip, various velocities, and Mach number) reflect the current initial conditions. A call is then made to `ls_model()`. This routine calls the user-supplied vehicle routines `inertias()`, `subsystems()`, `aero()`, `engine()`, and `gear()`, passing to them a value of 0 for time step and with the initialization flag non-zero, indicating a reset is requested. These user-supplied routines calculate the forces and moments for the current flight conditions, setting the appropriate values in the `generic_` data structure. A call is then made by `ls_step()` to the `ls_accel()` routine to sum the forces and moments and calculate appropriate initial accelerations at the vehicle center of gravity. `ls_aux()` is then called to calculate the appropriate accelerometer outputs. `ls_step()` then sets the local variable `dt = 0` and performs the normal state integration equations. Since `dt` is 0, the vehicle state is not updated; however, the past values of the integration filters become initialized to the appropriate initial condition values. Control flow then returns to `ls_init()`, which returns control to `main()`.

Continuing with the initialization process, `main()` calls `ls_record()` to record the initial time history data. The initial call to `ls_cockpit()` is then made, which initializes either the GL screen or the terminal display, depending on which interface routine was linked in at compile time - either the `curses` library routines to draw a simple instrument panel on the terminal, or the IRIS GL routines to draw an out-the-window and heads-up-display (HUD) presentation on a Silicon Graphics screen. A call is then made to `ls_sync()`, with `io_dt` passed as a parameter, which schedules an interval timer to signal `SIGALRM` on timer expiration.

The real-time loop portion of the program is then entered. This consists of `multiloop` number of passes to `ls_loop()`. `ls_loop()` calls the following sequence: `ls_step()`, which advances the simulation one `dt` in simulated time to a new state; `ls_aux()` which calculates the new flight conditions, based on the new state; `ls_model()`, which calculates new control positions as well as vehicle forces and moments at the reference point; and finally `ls_accel()`, which sums the forces and moments at the vehicle reference point, transfers them to the center of gravity, and then calculates the resulting accelerations. `ls_loop()` then returns control to `main()`.

`main()` then calls `ls_record()`, to record the current flight conditions, velocities, accelerations, and other parameters specified in the settings file. `main()` then makes a call to `ls_cockpit()` which refreshes the instrument panel display and gets new values for controls from the keyboard (or mouse, if GL is used). `ls_cockpit()` returns a non-zero integer if the user has signaled a desire to end the simulation. If `ls_cockpit()` returns zero, `ls_pause()` is called to await the arrival of the `SIGALRM` signal, which is caught and rescheduled, with command passing back to `main()` (see file `ls_sync.c`). If `Simtime` has exceeded the value of `endtime` or `ls_cockpit()` returned a non-zero value, the simulation calls the `ls_unsync()` and `ls_cockpit_exit()` routines, writes out any data files, calls `ls_put_settings()` to update the default settings file, and the program exits.

Concluding Remarks

This report describes how to implement, modify, and utilize a generic flight simulation software package on a UNIX-based computer. A description of each routine and all global variables are provided. The software is written entirely in ANSI C; listings of each routine are provided as well.

The structure of the code lends itself to pilot-in-the-loop operation on a sufficiently fast computer, and can be operated from a display terminal, a keyboard and mouse on a Silicon Graphics computer, or some modification, with an actual simulator cockpit. Time histories of selected parameters may be recorded in a variety of formats.

This software is patterned after similar FORTRAN routines used at the Manned Flight Simulator facility at the U.S. Navy's Naval Air Warfare Center/Aircraft Division, Patuxent River, Maryland. Those routines were themselves rewrites of older FORTRAN simulation routines that comprised a simulation architecture called BASIC used at NASA-Ames since the early 1970s.

The potential user is cautioned that results obtained from this software should be validated using conventional design methods. It is believed that equations of motion are implemented properly, but a full validation of LaRCsim against a benchmark simulation has not yet been performed. Simulated flight near either the North or South pole should be avoided, due to a singularity in the vehicle position calculations at either pole.

A copy of the latest version of this software is available upon request:

E. Bruce Jackson
MS 489
NASA Langley Research Center
Hampton, VA 23681-0001
e.b.jackson@larc.nasa.gov
(804) 864-4060

References

1. McFarland, Richard E., *A Standard Kinematic Model for Flight Simulation at NASA-Ames*, NASA CR-2497, January 1975.
2. ANSI/AIAA R-004-1992, *Recommended Practice: Atmospheric and Space Flight Vehicle Coordinate Systems*, February 1992.
3. Stevens, Brian L. and Lewis, Frank L., *Aircraft Control and Simulation*, Wiley and Sons, 1992, ISBN 0-471-61397-5.
4. Anon., *U. S. Standard Atmosphere, 1962.*
5. Anon., *Aeronautical Vest Pocket Handbook, 17th edition*, Pratt & Whitney Aircraft Group, Dec. 1977.
6. Halliday, David, and Resnick, Robert, *Fundamentals of Physics, Revised Printing*, Wiley and Sons, 1974, ISBN 0-471-34431-1.
7. Beyer, William H., editor, *CRC Standard Mathematical Tables, 28th edition*, CRC Press, Boca Raton, FL, 1987, ISBN 0-8493-0628-0.
8. Dowdy, M. C., Jackson, E. B., and Nichols, J. H., *Controls Analysis and Simulation Test Loop Environment (CASTLE) Programmer's Guide, Version 1.3*, TM 89-11, Naval Air Test Center, Patuxent River, MD, March 1989.
9. Zhu, J. *Exact Converions of Earth-Centered, Earth-Fixed Coordinates to Geodetic Coordinates.* J. Guidance, Control and Dynamics, vol. 16, no. 2, March-April 1993, p 389.
10. Morelli, Eugene A., *Nonlinear Aerodynamic Modeling using Multivariate Orthogonal Functions*, AIAA 93-3636, presented at the AIAA Atmospheric Flight Mechanics Conference, August 1993, Monterey, CA.
11. Maine, Richard E., *Manual for GetData Version 3.1, A FORTRAN Utility Program for Time History Data*, NASA TM-88288, October 1987.

Appendix A: LaRCsim Global Variables

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
Constants				
PI	Ratio of circumference to diameter of a circle	Macro definition	always positive	3.141592654
EQUATORIAL_RADIUS	Radius of the Earth at the equator	Macro definition	always positive	ft
RESQ	Square of radius of the Earth at the equator	Macro definition	always positive	ft^2
FP	Flattening parameter of oblate Earth	Macro definition	always positive	0.00335281
INVG	Inverse of sea level acceleration due to gravity	Macro definition	always positive	sec^2/ft
OMEGA_EARTH	Angular rotation velocity of the Earth	Macro definition	always positive	rad/sec
DEG_TO_RAD	Conversion factor, degrees to radians	Macro definition	always positive	deg/rad
RAD_TO_DEG	Conversion factor, radians to degrees	Macro definition	always positive	rad/deg
SEA_LEVEL_DENSITY	Atmospheric density at sea level at equator	Macro definition	always positive	slug/ft^3
Variables				
Time				
Simtime	Simulated time since beginning of current run			sec
Mass properties and geometry values				
Mass	Mass of simulated vehicle	Scalar	always positive	slugs
I_xx	Moment of inertia about X-body axis	Scalar	always positive	slug-ft^2
I_yy	Moment of inertia about Y-body axis	Scalar	always positive	slug-ft^2
I_zz	Moment of inertia about Y-body axis	Scalar	always positive	slug-ft^2
I_xz	Second moment of inertia in X-Z plane	Scalar	+Integral(x z dm)	slug-ft^2
D_pilot_rp_body_v[3]	Pilot offset from ref pt in body axis	3-element array	- -	ft
Dx_pilot	Pilot offset from ref pt in X body axis	Scalar	forward	ft
Dy_pilot	Pilot offset from ref pt in X body axis	Scalar	right	ft
Dz_pilot	Pilot offset from ref pt in X body axis	Scalar	down	ft
D_cg_rp_body_v[3]	Center of Gravity offset from ref pt in body axis	3-element array	- -	ft
Dx_cg	C.G. offset from ref pt in X body axis	Scalar	forward	ft
Dy_cg	C.G. offset from ref pt in Y body axis	Scalar	right	ft
Dz_cg	C.G. offset from ref pt in Z body axis	Scalar	down	ft

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
Forces				
F_body_total_v[3]	Total forces on body at ref pt in body axis	3-element array	- -	ft
F_X	Force along X-body axis at ref pt	Scalar	forward	ft
F_Y	Force along Y-body axis at ref pt	Scalar	right	ft
F_z	Force along Z-body axis at ref pt	Scalar	down	ft
F_local_total_v[3]	Total forces on body at ref pt in local axis	3-element array	- -	lbf
F_north	Northward force at ref pt	Scalar	north	lbf
F_east	Eastward force at ref pt	Scalar	east	lbf
F_down	Southward force at ref pt	Scalar	down	lbf
F_aero_v[3]	Aerodynamic forces on body at ref pt in body axis	3-element array	- -	lbf
F_X_aero	Aero force along X-body axis at ref pt	Scalar	forward	lbf
F_Y_aero	Aero force along Y-body axis at ref pt	Scalar	right	lbf
F_Z_aero	Aero force along Z-body axis at ref pt	Scalar	down	lbf
F_engine_v[3]	Engine forces on body at ref pt in body axis	3-element array	- -	lbf
F_X_engine	Engine force along X-body axis at ref pt	Scalar	forward	lbf
F_Y_engine	Engine force along Y-body axis at ref pt	Scalar	right	lbf
F_Z_engine	Engine force along Z-body axis at ref pt	Scalar	down	lbf
F_gear_v[3]	Landing gear forces on body at ref pt in body axis	3-element array	- -	lbf
F_X_gear	Gear force along X-body axis at ref pt	Scalar	forward	lbf
F_Y_gear	Gear force along Y-body axis at ref pt	Scalar	right	lbf
F_Z_gear	Gear force along Z-body axis at ref pt	Scalar	down	lbf

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
	Moments			
M_total_rp_v[3]	Total moments on body at ref pt measured around body axes	3-element array	- -	ft-lb
M_l_rp	Total moments on body at ref pt about X-body axis	Scalar	right wing down	ft-lb
M_m_rp	Total moments on body at ref pt about Y-body axis	Scalar	Nose up	ft-lb
M_n_rp	Total moments on body at ref pt about Z-body axis	Scalar	Nose left	ft-lb
M_total_cg_v[3]	Total moments on body at ref pt measured around body axes	3-element array	- -	ft-lb
M_l_cg	Total moments on body at ref pt about X-body axis	Scalar	right wing down	ft-lb
M_m_cg	Total moments on body at ref pt about Y-body axis	Scalar	Nose up	ft-lb
M_n_cg	Total moments on body at ref pt about Z-body axis	Scalar	Nose left	ft-lb
M_aero_v[3]	Aerodynamic moments on body at ref pt measured around body axes	3-element array	- -	ft-lb
M_l_aero	Aerodynamic moments on body at ref pt about X-body axis	Scalar	right wing down	ft-lb
M_m_aero	Aerodynamic moments on body at ref pt about Y-body axis	Scalar	Nose up	ft-lb
M_n_aero	Aerodynamic moments on body at ref pt about Z-body axis	Scalar	Nose left	ft-lb
M_engine_v[3]	Propulsion system moments on body at ref pt measured around body axes	3-element array	- -	ft-lb
M_l_engine	Propulsion system moments on body at ref pt about X-body axis	Scalar	right wing down	ft-lb
M_m_engine	Propulsion system moments on body at ref pt about Y-body axis	Scalar	Nose up	ft-lb
M_n_engine	Propulsion system moments on body at ref pt about Z-body axis	Scalar	Nose left	ft-lb
M_gear_v[3]	Landing gear moments on body at ref pt measured around body axes	3-element array	- -	ft-lb
M_l_gear	Landing gear moments on body at ref pt about X-body axis	Scalar	right wing down	ft-lb
M_m_gear	Landing gear moments on body at ref pt about Y-body axis	Scalar	Nose up	ft-lb
M_n_gear	Landing gear moments on body at ref pt about Z-body axis	Scalar	Nose left	ft-lb

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
	Accelerations			
V_dot_local_v[3]	Inertial acceleration of center of gravity measured in local axes	3-element array	- -	ft/sec^2
V_dot_north	Inertial acceleration of center of gravity measured in local North axis	Scalar	north	ft/sec^2
V_dot_east	Inertial acceleration of center of gravity measured in local East axis	Scalar	east	ft/sec^2
V_dot_down	Inertial acceleration of center of gravity measured in local down axis	Scalar	down	ft/sec^2
V_dot_body_v[3]	Inertial acceleration of ?? measured in body axes	3-element array	- -	ft/sec^2
U_dot_body	Inertial acceleration of ?? measured in body X axis	Scalar	forward	ft/sec^2
V_dot_body	Inertial acceleration of ?? measured in body Y axis	Scalar	right	ft/sec^2
W_dot_body	Inertial acceleration of ?? measured in body Z axis	Scalar	down	ft/sec^2
A_cg_body_v[3]	Inertial acceleration of center of gravity measured in body axes	3-element array	- -	ft/sec^2
A_X_cg	Inertial acceleration of center of gravity measured in body X axis	Scalar	forward	ft/sec^2
A_Y_cg	Inertial acceleration of center of gravity measured in body Y axis	Scalar	right	ft/sec^2
A_Z_cg	Inertial acceleration of center of gravity measured in body Z axis	Scalar	down	ft/sec^2
A_pilot_body_v[3]	Inertial acceleration of pilot station measured in body axes	3-element array	- -	ft/sec^2
A_X_pilot	Inertial acceleration of pilot station measured in body X axis	Scalar	forward	ft/sec^2
A_Y_pilot	Inertial acceleration of pilot station measured in body Y axis	Scalar	right	ft/sec^2
A_Z_pilot	Inertial acceleration of pilot station measured in body Z axis	Scalar	down	ft/sec^2
N_cg_body_v[3]	Inertial acceleration of center of gravity measured in body axes	3-element array	- -	g units
N_X_cg	Inertial acceleration of center of gravity measured in body X axis	Scalar	forward	g units
N_Y_cg	Inertial acceleration of center of gravity measured in body Y axis	Scalar	right	g units
N_Z_cg	Inertial acceleration of center of gravity measured in body Z axis	Scalar	down	g units
N_pilot_body_v[3]	Inertial acceleration of pilot station measured in body axes	3-element array	- -	g units
N_X_pilot	Inertial acceleration of pilot station measured in body X axis	Scalar	forward	g units
N_Y_pilot	Inertial acceleration of pilot station measured in body Y axis	Scalar	right	g units
N_Z_pilot	Inertial acceleration of pilot station measured in body Z axis	Scalar	down	g units
Omega_dot_body_v[3]	Angular acceleration of vehicle relative to local frame about center of gravity in body axe	3-element array	- -	rad/s^2
P_dot_body	Angular acceleration of vehicle relative to local frame about center of gravity in X body ax	Scalar	rt wing down	rad/s^2
Q_dot_body	Angular acceleration of vehicle relative to local frame about center of gravity in Y body ax	Scalar	nose up	rad/s^2
R_dot_body	Angular acceleration of vehicle relative to local frame about center of gravity in Z body ax	Scalar	nose right	rad/s^2

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
Velocities				
V_local_v[3]	Inertial velocity of center of gravity in local axes	3-element array	- -	ft/s
V_north	Inertial velocity of center of gravity in local North axis	Scalar	north	ft/s
V_east	Inertial velocity of center of gravity in local East axis	Scalar	east	ft/s
V_down	Inertial velocity of center of gravity in local down axis	Scalar	down	ft/s
V_local_rel_ground_v[3]	Velocity of center of gravity relative to earth surface in local axes	3-element array	- -	ft/s
V_north_rel_ground	Velocity of center of gravity relative to earth surface in local North axis	Scalar	north	ft/s
V_east_rel_ground	Velocity of center of gravity relative to earth surface in local east axis	Scalar	east	ft/s
V_down_rel_ground	Velocity of center of gravity relative to earth surface in local down axis	Scalar	down	ft/s
V_local_airmass_v[3]	Inertial steady-state velocity of airmass in local axes	3-element array	- -	ft/s
V_north_airmass	Inertial steady-state velocity of airmass in local North axis	Scalar	north	ft/s
V_east_airmass	Inertial steady-state velocity of airmass in local East axis	Scalar	east	ft/s
V_down_airmass	Inertial steady-state velocity of airmass in local down axis	Scalar	down	ft/s
V_local_rel_airmass_v[3	Velocity of center of gravity relative to local airmass in local axes	3-element array	- -	ft/s
V_north_rel_airmass	Velocity of center of gravity relative to local airmass in local North axis	Scalar	north	ft/s
V_east_rel_airmass	Velocity of center of gravity relative to local airmass in local East axis	Scalar	east	ft/s
V_down_rel_airmass	Velocity of center of gravity relative to local airmass in local down axis	Scalar	down	ft/s
V_body_gust_v[3]	Gust velocity in body axes	3-element array	- -	ft/s
U_gust	Gust velocity in X-body axes	Scalar	forward	ft/s
V_gust	Gust velocity in Y-body axes	Scalar	right	ft/s
W_gust	Gust velocity in Z-body axes	Scalar	down	ft/s
V_wind_body_v[3]	Velocity of center of gravity relative to local airmass in body axes	3-element array	- -	ft/s
U_body	Velocity of center of gravity relative to local airmass in X-body axis	Scalar	forward	ft/s
V_body	Velocity of center of gravity relative to local airmass in Y-body axis	Scalar	right	ft/s
W_body	Velocity of center of gravity relative to local airmass in Z-body axis	Scalar	down	ft/s
V_rel_wind	Velocity relative to airmass	Scalar	always positive	ft/s
V_true_knots	True airspeed in knots	Scalar	always positive	nm/hr
V_rel_ground	Velocity relative to earth's surface	Scalar	always positive	ft/s
V_inertial	Inertial velocity	Scalar	always positive	ft/s
V_ground_speed	Velocity at right angles to local vertical	Scalar	always positive	ft/s
V_equiv	Equivalent airspeed	Scalar	always positive	ft/s
V_equiv_knots	Equivalent airspeed, knots	Scalar	always positive	nm/hr
V_calibrated	Calibrated airspeed	Scalar	always positive	ft/s
V_calibrated_kts	Calibrated airspeed, knots	Scalar	always positive	nm/hr

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
	Velocities, cont'd			
Omega_body_v[3]	Inertial rotational rate of the body axis frame	3-element array	- -	rad/s
P_body	Inertial rotational rate of the body X-axis	Scalar	rt wing down	rad/s
Q_body	Inertial rotational rate of the body Y-axis	Scalar	nose up	rad/s
R_body	Inertial rotational rate of the body Z-axis	Scalar	nose right	rad/s
Omega_local_v[3]	Inertial rotational rate of the local axis frame	3-element array	- -	rad/s
P_local	Inertial rotational rate of the local axis frame about the body X-axis	Scalar	rt wing down	rad/s
Q_local	Inertial rotational rate of the local axis frame about the body Y-axis	Scalar	nose up	rad/s
R_local	Inertial rotational rate of the local axis frame about the body Z-axis	Scalar	nose right	rad/s
Omega_total_v[3]	Rotational rate of the body axis frame relative to the local axis frame	3-element array	- -	rad/s
P_total	Rotational rate of the body axis frame relative to the local axis frame about the body X-ax	Scalar	rt wing down	rad/s
Q_total	Rotational rate of the body axis frame relative to the local axis frame about the body Y-ax	Scalar	nose up	rad/s
R_total	Rotational rate of the body axis frame relative to the local axis frame about the body Z-ax	Scalar	nose right	rad/s
Euler_rates_v[3]	Rotational rate of the body axis frame relative to the local axis frame, in Euler angles	3-element array	- -	rad/s
Phi_dot	Rotational rate of the body axis frame about the local X-axis	Scalar	rt wing down	rad/s
Theta_dot	Rotational rate of the body axis frame about the local Y-axis	Scalar	nose up	rad/s
Psi_dot	Rotational rate of the body axis frame about the local Z-axis	Scalar	nose right	rad/s
Geocentric_rates_v[3]	Rotational rate of the body axis frame relative to the inertial frame	3-element array	- -	- -
Latitude_dot	Rate of change of geocentric latitude angle	Scalar	westward	rad/s
Longitude_dot	Rate of change of geocentric longitude angle	Scalar	northward	rad/s
Radius_dot	Rate of change of radius from center of inertial frame	Scalar	outward	ft/s
	Positions			
Geocentric_position_v[3	Geocentric position of vehicle's center of gravity	3-element array	- -	- -
Lat_geocentric	Geocentric latitude of vehicle's center of gravity	Scalar	westward	rad
Lon_geocentric	Geocentric longitude of vehicle's center of gravity	Scalar	northward	rad
Radius_to_vehicle	Radius to vehicle's center of gravity from inertial frame	Scalar	outward	ft
Geodetic_position_v[3]	Geodetic position of vehicle's center of gravity	3-element array	- -	- -
Latitude	Geodetic latitude of vehicle's center of gravity	Scalar	westward	rad
Longitude	Geodetic longitude of vehicle's center of gravity	Scalar	northward	rad
Altitude	Height of vehicle's center of gravity above reference ellipsoid	Scalar	outward	ft
Euler_angles_v[3]	Vehicle's angular attitude relative to local frame	3-element array	- -	rad
Phi	Roll angle	Scalar	rt wing down	rad
Theta	Pitch angle	Scalar	nose up	rad
Psi	Heading angle	Scalar	nose right	rad

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
Miscellaneous quantities				
T_local_to_body_m[3][3]	Transformation matrix L to B	3 by 3 matrix	- -	- -
T_local_to_body_11	Transformation matrix element	Scalar	- -	- -
T_local_to_body_12	Transformation matrix element	Scalar	- -	- -
T_local_to_body_13	Transformation matrix element	Scalar	- -	- -
T_local_to_body_21	Transformation matrix element	Scalar	- -	- -
T_local_to_body_22	Transformation matrix element	Scalar	- -	- -
T_local_to_body_23	Transformation matrix element	Scalar	- -	- -
T_local_to_body_31	Transformation matrix element	Scalar	- -	- -
T_local_to_body_32	Transformation matrix element	Scalar	- -	- -
T_local_to_body_33	Transformation matrix element	Scalar	- -	- -
Gravity	Acceleration due to earth's gravity	Scalar	down	ft/s^s
Centrifugal_relief	Centrifugal acceleration due to near-orbital speed	Scalar	up	ft/s^2
Alpha	Free-stream angle of attack	Scalar	nose up	deg
Beta	Free-stream angle of sideslip	Scalar	nose left	deg
Alpha_dot	Time rate of change of free-stream angle of attack	Scalar	nose up	deg/s
Beta_dot	Time rate of change of free-stream angle of sideslip	Scalar	nose left	deg/s
Cos_alpha	Cosine of free-stream angle of attack	Scalar	nose up	- -
Sin_alpha	Sine of free-stream angle of attack	Scalar	nose up	- -
Cos_beta	Cosine of free-stream angle of sideslip	Scalar	nose left	- -
Sin_beta	Sine of free-stream angle of sideslip	Scalar	nose left	- -
Cos_phi	Cosine of bank angle	Scalar	rt wing down	- -
Sin_phi	Sine of bank angle	Scalar	rt wing down	- -
Cos_theta	Cosine of pitch angle	Scalar	nose up	- -
Sin_theta	Sine of pitch angle	Scalar	nose up	- -
Cos_psi	Cosine of heading angle	Scalar	nose right	- -
Sin_psi	Sine of heading angle	Scalar	nose right	- -
Gamma_vert_rad	Vertical flight path angle in local frame	Scalar	climb	rad
Gamma_horiz_rad	Horizontal flight path, or track, angle in local frame	Scalar	clockwise from north	rad
Sigma	Ratio of free-stream density to sea-level reference density	Scalar	always positive	- -
Density	Atmospheric density (free-stream flight conditions)	Scalar	always positive	slug/ft^3
V_sound	Speed of sound (free-stream flight conditions)	Scalar	always positive	ft/s
Mach_number	Free-stream mach number	Scalar	always positive	- -
Static_pressure	Static pressure	Scalar	always positive	lb/ft^2
Total_pressure	Total pressure	Scalar	always positive	lb/ft^2
Impact_pressure	Impact pressure	Scalar	always positive	lb/ft^2
Dynamic_pressure	Dynamic pressure	Scalar	always positive	lb/ft^2

Macro or variable name	Variable Description	Data type	Sign convention (positive when..)	Units of Measure
Miscellaneous quantities, cont'd				
Static_temperature	Static temperature	Scalar	always positive	R
Total_temperature	Total temperature	Scalar	always positive	R
Sea_level_radius	Radius from earth center to local plumb sea level	Scalar	outward	ft
Earth_position_angle	Amount of rotation of the earth since reference time	Scalar	from ref time	rad
Runway_altitude	Height of runway threshold above local plumb sea level (geodetic)	Scalar	up	ft
Runway_latitude	Geodetic latitude of runway threshold	Scalar	northward	rad
Runway_longitude	Geodetic longitude of runway threshold	Scalar	westward	rad
Runway_heading	Runway heading	Scalar	clockwise from north	rad
Radius_to_rwy	Radius from earth center to runway threshold point	Scalar	outward	ft
D_cg_rwy_local_v[3];	Location of center of gravity relative to runway threshold in local frame	3-element array	- -	ft
D_cg_north_of_rwy	Distance of center of gravity northward from runway threshold	Scalar	northward	ft
D_cg_east_of_rwy	Distance of center of gravity eastward from runway threshold	Scalar	eastward	ft
D_cg_above_rwy	Height of center of gravity above runway threshold	Scalar	up	ft
D_cg_rwy_rwy_v[3]	Location of center of gravity relative to runway threshold in runway frame	3-element array	- -	ft
X_cg_rwy	Distance of center of gravity along runway centerline	Scalar	beyond threshold	ft
Y_cg_rwy	Distance of center of gravity right of runway centerline	Scalar	right of CL	ft
H_cg_rwy	Height of center of gravity above runway threshold	Scalar	up	ft
D_pilot_rwy_local_v[3]	Location of pilot's eyepoint relative to runway threshold in local frame	3-element array	- -	ft
D_pilot_north_of_rwy	Distance of pilot's eyepoint northward form runway threshold	Scalar	northward	ft
D_pilot_east_of_rwy	Distance of pilot's eyepoint eastward from runway threshold	Scalar	eastward	ft
D_pilot_above_rwy	Height of pilot's eyepoint above runway threshold	Scalar	up	ft
D_pilot_rwy_rwy_v[3]	Location of pilot's eyepoint relative to runway threshold in runway frame	3-element array	- -	ft
X_pilot_rwy	Distance of pilot's eyepoint along runway centerline	Scalar	beyond threshold	ft
Y_pilot_rwy	Distance of pilot's eyepoint right of runway centerline	Scalar	right of CL	ft
Z_pilot_rwy	Height of pilot's eyepoint above runway threshold	Scalar	up	ft

Appendix B: Source Code Listings

```
# Makefile for LaRCSIM - AGCB internal version
#
# 930909 EBJ
#
#$Header: /aces/larcsim/dev/RCS/Makefile,v 1.22 1995/04/07 01:40:47 bjax Exp $
#
#$Log: Makefile,v $
# Revision 1.22  1995/04/07  01:40:47  bjax
# Updated dependencies of ls_sim_control.h
#
# Revision 1.21  1995/03/29  16:16:44  bjax
# Added calACES as program output; removed PVI/VISION stuff; renamed ifsun to ifterm.
#
# Revision 1.20  1995/03/08  12:32:49  bjax
# Big boo-boo! Had all the header file dependencies set for _source_ files,
# not _object_ files. Boy, am I dumb. OK, this version fixes that, so changes
# to header files cause affected object files to be recompiled. Whew!  EBJ
#
# Revision 1.19  1995/03/06  18:44:53  bjax
# Added ls_settings object file to compilation list. EBJ
#
# Revision 1.18  1995/02/27  19:51:51  bjax
# Added ls_trim & ls_matrix routines to support trimming. EJB
#
# Revision 1.17  1994/12/02  17:22:03  bjax
# Removed dependency on VISION/PVI stuff; links are messed up due to moving files
# from /usr/people/pvi to /hewitt/pvi, and now that Alan Dare has gone, I don't
# know an easy way to change 'em to the right place.  Commented out appropriate
# lines in the makefile and added modified lines immediately following 'em. EBJ
#
# Revision 1.16  1994/07/12  21:37:00  bjax
# Added dependency of ls_ACES.c upon ls_ACES.h.
#
# Revision 1.15  1994/05/13  20:44:04  bjax
# Renamed ls_main to LaRCsim.
#
# Revision 1.14  1994/05/10  15:15:40  bjax
# Change from cmp2 to ascl.
#
# Revision 1.13  1994/05/10  15:06:05  bjax
# Really added ls_writeascl this time!
#
# Revision 1.12  1994/05/10  15:04:22  bjax
# Added ls_writeascl and ls_writetab routines.
#
# Revision 1.11  1994/05/06  20:25:39  bjax
# Added dependency of ls_record.c on ls_sym.h
#
# Revision 1.10  1994/02/04  13:21:02  bjax
# Rearranged to separate the interface routines, which have redundant
# routine names, from being placed in the libls.a archive. This will
# require all simulation Makefiles to explicitly select which interface
# to link with, and will get rid of the linker warning when linking in
# a customized interface routine.
#
# Revision 1.9  1994/01/11  19:25:35  bjax
# Updated dependencies.
#
# Revision 1.8  1994/01/05  19:59:12  bjax
# Changed name of cockpit routine to ls_ifgl.c for consistency
# (Was ls_glcockpit.c).
#
# Revision 1.7  1993/12/14  21:12:16  bjax
# Added -I$(LARCSIM) flag to CFLAG macro defn.  EBJ
#
# Revision 1.6  1993/12/14  21:05:25  bjax
# Changed path to vmic/adc to ../vmic/adc to reflect new structure.
#
# Revision 1.5  1993/12/04  12:39:20  bjax
# Added readstick dependency.
#
# Revision 1.4.1.1  1993/09/09  22:44:07  bjax
# Added ls_readstick module dependencies.
#
# Revision 1.4  1993/09/09  22:41:06  bjax
# This branch includes selectable 'cockpit' (for A/D interface).
#


SHELL = /usr/bin/sh

OBJECTS =       \
LaRCsim.o       \
ls_ACES.o       \
ls_accel.o      \
atmos_62.o      \
ls_aux.o        \
ls_err.o        \
ls_funcgen.o    \
ls_geodesy.o    \
ls_gravity.o    \
ls_matrix.o     \
ls_model.o      \
ls_record.o     \
ls_settings.o   \
ls_step.o       \
ls_sym.o        \
ls_sync.o       \
ls_trim.o       \
ls_writeav.o    \
ls_writemat.o   \
ls_writetab.o   \
ls_writeascl.o  \
ls_init.o       \
default_model_routines.o

IF_OBJECTS = ls_ifgl.o ls_ifterm.o

CC = cc
CFLAGS = -g -I$(LARCSIM)
AR = ar
ARFLAGS = -r
CO = co
COFLAGS = -f

make: libls.a $(IF_OBJECTS) calACES

libls.a : libls.a($(OBJECTS))

$(OBJECTS): ls_types.h ls_constants.h $(*.c)

calACES: cal.c
	cc -g cal.c -lgl -lc -o calACES

cal.o: cal.c ls_types.h ls_ACES.h ls_cockpit.h


#additional dependencies:
ls_trim.o ls_sym.o ls_record.o: ls_sym.h
ls_trim.o ls_matrix.o: ls_matrix.h
```

```
ls_funcgen.o:   ls_funcgen.h
ls_err.o  ls_funcgen.o: ls_err.h
ls_ACES.o ls_ifgl.o:    ls_cockpit.h ls_ACES.h
LaRCsim.o ls_ifgl.o ls_ifterm.o ls_record.o ls_settings.c ls_step.o ls_sync.o \
    ls_writeascl.o ls_writeav.o ls_writemat.o ls_writetab.o : ls_sim_control.h
LaRCsim.o ls_record.o ls_writetab.o ls_writeascl.o ls_writeav.o ls_writemat.o:  ls_tape.
h ls_sym.h
ls_trim.o ls_accel.o ls_aux.o ls_ifgl.o LaRCsim.o ls_record.o ls_step.o:        ls_gener
ic.h

ls_eom.h ls_tape.h ls_err.h ls_funcgen.h ls_sym.h ls_matrix.h : $(RCS/@:.h=.h,v)
        -mv $(@) $(@).sav
        $(CO) $(COFLAGS) $(@)

#ls_ifpvi.h : ls_types.h ls_generic.h ls_sim_control.h ls_cockpit.h \
#       $(VISIONPATH)/include/realtime.h

count :
        co RCS/*
        cat *.c | cat - *.h | wc -l

cleanup:
        -rm libls.a
        -rm *.o
        -rm *.bak
        -rm *~
```

```
/***************************************************************************

        TITLE:      ls_ACES.h

----------------------------------------------------------------------------

        FUNCTION:       Include file for routines that use the ACES cockpit

----------------------------------------------------------------------------

        MODULE STATUS:  incomplete

----------------------------------------------------------------------------

        GENEALOGY:      Created 930215 by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    E. B. Jackson

        CODED BY:       E. B. Jackson

        MAINTAINED BY:  E. B. Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_ACES.h,v 1.4 1995/02/28 20:35:40 bjax Stab $
$Log: ls_ACES.h,v $
 * Revision 1.4  1995/02/28  20:35:40  bjax
 * Changed name of GEAR_SEL_DOWN to GEAR_SEL_UP to reflect correct
 * sense of switch. EBJ
 *
 * Revision 1.3  1994/07/12  21:36:13  bjax
 * Fixed error between SB_EXTEND and SB_RETRACT
 * bits.
 *
 * Revision 1.2  1994/04/11  20:41:17  bjax
 * Added rudder pedal calibration data & scaling to support THRUSTMASTER!
 *
 * Revision 1.1  1994/02/15  20:37:09  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#define VMIC3114_CONTROL_ADDR   0x00006000
#define VMIC3114_CONTROL_LENGTH 0x20
#define VMIC3114_CONTROL_BUS_DEVICE "/dev/vme/vme0a16n"

#define VMIC3114_DATA_ADDR   0x00C00000
#define VMIC3114_DATA_LENGTH 0x00040000
#define VMIC3114_DATA_BUS_DEVICE "/dev/vme/vme0a24n"

#define LANDING_GEAR_UP      0x0010
#define SPEEDBRAKE_EXTEND    0x0040
#define SPEEDBRAKE_RETRACT   0x0080
#define LEFT_PUSH_BUTTON     0x0100
#define RIGHT_PUSH_BUTTON    0x0200
#define SECOND_TRIG_BUTTON   0x0400
#define FIRST_TRIG_BUTTON    0x0800
#define RIGHT_TRIM           0x1000
#define FWD_TRIM             0x2000
#define AFT_TRIM             0x4000
#define LEFT_TRIM            0x8000


    typedef struct
    {
        float thraft[4];    /* aft throttle stop A/D values     */
        float thrfwd[4];    /* fwd throttle stop A/D values     */
        float stkaft;       /* aft long stick stop A/D value    */
        float stklg0;       /* center long stick stop A/D value */
        float stkfwd;       /* fwd long stick stop A/D value    */
        float stklft;       /* left lat stick stop A/D value    */
        float stklt0;       /* center lat stick stop A/D value  */
        float stkrgt;       /* right lat stick stop A/D value   */
        float pedlft;       /* left rudder pedal stop A/D value */
        float pedctr;       /* center rudder pedal A/D value    */
        float pedrgt;       /* right rudder pedal stop A/D val  */
    } CALIBRATION_DATA;
```

```
/***************************************************************************

        TITLE:          ls_cockpit.h

----------------------------------------------------------------------------

        FUNCTION:       Header for cockpit IO

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 20 DEC 93 by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    E. B. Jackson

        CODED BY:       E. B. Jackson

        MAINTAINED BY:  E. B. Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950314  Added "throttle_pct" field to cockpit header for both
                display and trim purposes.                      EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_cockpit.h,v 1.3 1995/03/15 12:32:10 bjax Stab $
$Log: ls_cockpit.h,v $
 * Revision 1.3  1995/03/15  12:32:10  bjax
 * Added throttle_pct field.
 *
 * Revision 1.2  1995/02/28  20:37:02  bjax
 * Changed name of gear_sel_down switch to gear_sel_up to reflect
 * correct sense of switch.  EBJ
 *
 * Revision 1.1  1993/12/21  14:39:04  bjax
 * Initial revision
 *

--------------------------------------------------------------------------*/

typedef struct {
    float   long_stick, lat_stick, rudder_pedal;
    float   throttle[4];
    short   forward_trim, aft_trim, left_trim, right_trim;
    short   left_pb_on_stick, right_pb_on_stick, trig_pos_1, trig_pos_2;
    short   sb_extend, sb_retract, gear_sel_up;
    float   throttle_pct;
} COCKPIT;

extern COCKPIT cockpit_;

#define Left_button     cockpit_.left_pb_on_stick
#define Right_button    cockpit_.right_pb_on_stick
#define Rudder_pedal    cockpit_.rudder_pedal
#define Throttle        cockpit_.throttle
```



```
#define Throttle_pct    cockpit_.throttle_pct
#define First_trigger   cockpit_.trig_pos_1
#define Second_trigger  cockpit_.trig_pos_2
#define Long_control    cockpit_.long_stick
#define Lat_control     cockpit_.lat_stick
#define Fwd_trim        cockpit_.forward_trim
#define Aft_trim        cockpit_.aft_trim
#define Left_trim       cockpit_.left_trim
#define Right_trim      cockpit_.right_trim
#define SB_extend       cockpit_.sb_extend
#define SB_retract      cockpit_.sb_retract
#define Gear_sel_up     cockpit_.gear_sel_up
```

```
/***************************************************************************

        TITLE:  ls_constants.h

----------------------------------------------------------------------------

        FUNCTION:       LaRCSim constants definition header file

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 15 DEC 1993 by Bruce Jackson; was part of
                        old ls_eom.h header file

----------------------------------------------------------------------------

        DESIGNED BY:    B. Jackson

        CODED BY:       B. Jackson

        MAINTAINED BY:  guess who

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

----------------------------------------------------------------------------

        REFERENCES:

                [ 1]    McFarland, Richard E.: "A Standard Kinematic Model
                        for Flight Simulation at NASA-Ames", NASA CR-2497,
                        January 1975

                [ 2]    ANSI/AIAA R-004-1992 "Recommended Practice: Atmos-
                        pheric and Space Flight Vehicle Coordinate Systems",
                        February 1992

                [ 3]    Beyer, William H., editor: "CRC Standard Mathematical
                        Tables, 28th edition", CRC Press, Boca Raton, FL, 1987,
                        ISBN 0-8493-0628-0

                [ 4]    Dowdy, M. C.; Jackson, E. B.; and Nichols, J. H.:
                        "Controls Analysis and Simulation Test Loop Environ-
                        ment (CASTLE) Programmer's Guide, Version 1.3",
                        NATC TM 89-11, 30 March 1989.

                [ 5]    Halliday, David; and Resnick, Robert: "Fundamentals
                        of Physics, Revised Printing", Wiley and Sons, 1974.
                        ISBN 0-471-34431-1

                [ 6]    Anon: "U. S. Standard Atmosphere, 1962"

                [ 7]    Anon: "Aeronautical Vest Pocket Handbook, 17th edition",
                        Pratt & Whitney Aircraft Group, Dec. 1977

                [ 8]    Stevens, Brian L.; and Lewis, Frank L.: "Aircraft
                        Control and Simulation", Wiley and Sons, 1992.
                        ISBN 0-471-61397-5

--------------------------------------------------------------------------*/

#ifndef CONSTANTS

#define CONSTANTS -1

/* Define application-wide macros */

#define PATHNAME "LARCSIMPATH"
#ifndef NIL_POINTER
#define NIL_POINTER 0L
#endif

/* Define constants (note: many factors will need to change for other
        systems of measure)     */

/* Value of Pi from ref [3] */
#define PI 3.14159265358979323846264338327950288419716939967511

/* Value of earth radius from [8], ft */
#define EQUATORIAL_RADIUS 20925650.
#define RESQ 437882827922500.

/* Value of earth flattening parameter from ref [8]

        Note: FP = f
              E  = 1-f
              EPS = sqrt(1-(1-f)^2)                     */

#define FP      .003352813178
#define E    .996647186
#define EPS .081819221
#define INVG .031080997

/* linear velocity of earth at equator from w*R; w=2pi/24 hrs, in ft/sec */
#define OMEGA_EARTH .00007272205217

/* miscellaneous units conversions (ref [7]) */
#define V_TO_KNOTS      0.5921
#define DEG_TO_RAD      0.017453292
#define RAD_TO_DEG      57.29577951
#define FT_TO_METERS    0.3048
#define METERS_TO_FT    3.2808
#define K_TO_R          1.8
#define R_TO_K          0.55555556
#define NSM_TO_PSF      0.02088547
#define PSF_TO_NSM      47.8801826
#define KCM_TO_SCF      0.00194106
#define SCF_TO_KCM      515.183616


/* ENGLISH Atmospheric reference properties [6] */
#define SEA_LEVEL_DENSITY       0.002376888

#endif

/*------------------------ end of ls_constants.h ------------------------*/
```

```
/***************************************************************************

        TITLE:          ls_err.h

----------------------------------------------------------------------------

        FUNCTION:       Simulation error reporting structure

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      Written 9112 by B. Jackson for MATLAB Mex file
                        table lookup algorithms, installed 930319 as part
                        of LaRCSim software.

----------------------------------------------------------------------------

        DESIGNED BY:    B. Jackson

        CODED BY:       B. Jackson

        MAINTAINED BY:  B. Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_err.h,v 1.1 1993/03/19 07:01:54 bjax Stab $
$Log: ls_err.h,v $
 * Revision 1.1  1993/03/19  07:01:54  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#define ERROR_STRING_LENGTH     256

typedef struct
{
        enum    ( info, warning, fatal ) severity;
        int     code;
        char    *strg1;
        char    *strg2;
        float   fp1, fp2, fp3;
        int     ip1, ip2, ip3;
} ERROR;


/*  Error code definitions */

#define E_NO_ERROR                0
#define E_DATA_INVALID           -1
#define E_FUNCGEN_INDEX_ERROR    -2
```

```
/***************************************************************************

        TITLE:          ls_funcgen.h

----------------------------------------------------------------------------

        FUNCTION:       Function generation routines header file

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      Developed circa 1991 by E. B. Jackson; installed
                        as part of LaRCSim 930319.

----------------------------------------------------------------------------

        DESIGNED BY:    B. Jackson

        CODED BY:       B. Jackson

        MAINTAINED BY:  B. Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_funcgen.h,v 1.1 1993/03/19 07:02:32 bjax Stab $
$Log: ls_funcgen.h,v $
 * Revision 1.1  1993/03/19  07:02:32  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#define MAX_DIMENSION  6
#define MAX_LENGTH  63
#define MAX_DATA_NAME_LENGTH  15

typedef double DATA;

typedef struct
{
        long    index[ MAX_DIMENSION ];
        float   index_and_weight[ MAX_DIMENSION ];
}       ARG_LIST;

typedef struct
{
        char    name[ MAX_DATA_NAME_LENGTH ];
        int             length;
        DATA    bkPts[ MAX_LENGTH ];
} BREAKPOINTS;

typedef struct
{
        char    name[ MAX_DATA_NAME_LENGTH ];
        int             dim;
        int             length[MAX_DIMENSION];
        DATA    *pts;
}       FUNC_DATA;

typedef struct
{
        char            name[ MAX_DATA_NAME_LENGTH ];
        FUNC_DATA       *ptr_to_data;
        BREAKPOINTS     *bkPtList[ MAX_DIMENSION ];
        float           latest_index_and_weights[ MAX_DIMENSION ];
        DATA            latest_bkpt_value[ MAX_DIMENSION ];
}       NONLINEAR_FUNCTION;

float normalize_bkpt( NONLINEAR_FUNCTION *nlfunct,int dim, DATA value);

DATA funcgen( NONLINEAR_FUNCTION *func_ptr, ARG_LIST *arg_list, int dim);
```

```
/***************************************************************************

	TITLE:	ls_generic.h

----------------------------------------------------------------------------

	FUNCTION:	LaRCSim generic parameters header file

----------------------------------------------------------------------------

	MODULE STATUS:	developmental

----------------------------------------------------------------------------

	GENEALOGY:	Created 15 DEC 1993 by Bruce Jackson;
			was part of old ls_eom.h header

----------------------------------------------------------------------------

	DESIGNED BY:	B. Jackson

	CODED BY:	B. Jackson

	MAINTAINED BY:	guess who

----------------------------------------------------------------------------

	MODIFICATION HISTORY:

	DATE	PURPOSE						BY

----------------------------------------------------------------------------

	REFERENCES:

		[ 1]	McFarland, Richard E.: "A Standard Kinematic Model
			for Flight Simulation at NASA-Ames", NASA CR-2497,
			January 1975

		[ 2]	ANSI/AIAA R-004-1992 "Recommended Practice: Atmos-
			pheric and Space Flight Vehicle Coordinate Systems",
			February 1992

		[ 3]	Beyer, William H., editor: "CRC Standard Mathematical
			Tables, 28th edition", CRC Press, Boca Raton, FL, 1987,
			ISBN 0-8493-0628-0

		[ 4]	Dowdy, M. C.; Jackson, E. B.; and Nichols, J. H.:
			"Controls Analysis and Simulation Test Loop Environ-
			ment (CASTLE) Programmer's Guide, Version 1.3",
			NATC TM 89-11, 30 March 1989.

		[ 5]	Halliday, David; and Resnick, Robert: "Fundamentals
			of Physics, Revised Printing", Wiley and Sons, 1974.
			ISBN 0-471-34431-1

		[ 6]	Anon: "U. S. Standard Atmosphere, 1962"

		[ 7]	Anon: "Aeronautical Vest Pocket Handbook, 17th edition",
			Pratt & Whitney Aircraft Group, Dec. 1977

		[ 8]	Stevens, Brian L.; and Lewis, Frank L.: "Aircraft
			Control and Simulation", Wiley and Sons, 1992.
			ISBN 0-471-61397-5

---------------------------------------------------------------------------*/

typedef struct {

/*================== Mass properties and geometry values ==================*/

    DATA    mass, i_xx, i_yy, i_zz, i_xz;        /* Inertias */
#define Mass                    generic_.mass
#define I_xx                    generic_.i_xx
#define I_yy                    generic_.i_yy
#define I_zz                    generic_.i_zz
#define I_xz                    generic_.i_xz

    VECTOR_3    d_pilot_rp_body_v;      /* Pilot location rel to ref pt */
#define D_pilot_rp_body_v       generic_.d_pilot_rp_body_v
#define Dx_pilot                generic_.d_pilot_rp_body_v[0]
#define Dy_pilot                generic_.d_pilot_rp_body_v[1]
#define Dz_pilot                generic_.d_pilot_rp_body_v[2]

    VECTOR_3    d_cg_rp_body_v; /* CG position w.r.t. ref. point */
#define D_cg_rp_body_v          generic_.d_cg_rp_body_v
#define Dx_cg                   generic_.d_cg_rp_body_v[0]
#define Dy_cg                   generic_.d_cg_rp_body_v[1]
#define Dz_cg                   generic_.d_cg_rp_body_v[2]

/*============================== Forces ===============================*/

    VECTOR_3    f_body_total_v;
#define F_body_total_v          generic_.f_body_total_v
#define F_X                     generic_.f_body_total_v[0]
#define F_Y                     generic_.f_body_total_v[1]
#define F_Z                     generic_.f_body_total_v[2]

    VECTOR_3    f_local_total_v;
#define F_local_total_v         generic_.f_local_total_v
#define F_north                 generic_.f_local_total_v[0]
#define F_east                  generic_.f_local_total_v[1]
#define F_down                  generic_.f_local_total_v[2]

    VECTOR_3    f_aero_v;
#define F_aero_v                generic_.f_aero_v
#define F_X_aero                generic_.f_aero_v[0]
#define F_Y_aero                generic_.f_aero_v[1]
#define F_Z_aero                generic_.f_aero_v[2]

    VECTOR_3    f_engine_v;
#define F_engine_v              generic_.f_engine_v
#define F_X_engine              generic_.f_engine_v[0]
#define F_Y_engine              generic_.f_engine_v[1]
#define F_Z_engine              generic_.f_engine_v[2]

    VECTOR_3    f_gear_v;
#define F_gear_v                generic_.f_gear_v
#define F_X_gear                generic_.f_gear_v[0]
#define F_Y_gear                generic_.f_gear_v[1]
#define F_Z_gear                generic_.f_gear_v[2]

/*============================== Moments ===============================*/

    VECTOR_3    m_total_rp_v;
#define M_total_rp_v            generic_.m_total_rp_v
#define M_l_rp                  generic_.m_total_rp_v[0]
#define M_m_rp                  generic_.m_total_rp_v[1]
#define M_n_rp                  generic_.m_total_rp_v[2]
```

```
    VECTOR_3    m_total_cg_v;
#define M_total_cg_v            generic_.m_total_cg_v
#define M_l_cg                  generic_.m_total_cg_v[0]
#define M_m_cg                  generic_.m_total_cg_v[1]
#define M_n_cg                  generic_.m_total_cg_v[2]

    VECTOR_3    m_aero_v;
#define M_aero_v                generic_.m_aero_v
#define M_l_aero                generic_.m_aero_v[0]
#define M_m_aero                generic_.m_aero_v[1]
#define M_n_aero                generic_.m_aero_v[2]

    VECTOR_3    m_engine_v;
#define M_engine_v              generic_.m_engine_v
#define M_l_engine              generic_.m_engine_v[0]
#define M_m_engine              generic_.m_engine_v[1]
#define M_n_engine              generic_.m_engine_v[2]

    VECTOR_3    m_gear_v;
#define M_gear_v                generic_.m_gear_v
#define M_l_gear                generic_.m_gear_v[0]
#define M_m_gear                generic_.m_gear_v[1]
#define M_n_gear                generic_.m_gear_v[2]

/*============================= Accelerations ============================*/

    VECTOR_3    v_dot_local_v;
#define V_dot_local_v           generic_.v_dot_local_v
#define V_dot_north             generic_.v_dot_local_v[0]
#define V_dot_east              generic_.v_dot_local_v[1]
#define V_dot_down              generic_.v_dot_local_v[2]

    VECTOR_3    v_dot_body_v;
#define V_dot_body_v            generic_.v_dot_body_v
#define U_dot_body              generic_.v_dot_body_v[0]
#define V_dot_body              generic_.v_dot_body_v[1]
#define W_dot_body              generic_.v_dot_body_v[2]

    VECTOR_3    a_cg_body_v;
#define A_cg_body_v             generic_.a_cg_body_v
#define A_X_cg                  generic_.a_cg_body_v[0]
#define A_Y_cg                  generic_.a_cg_body_v[1]
#define A_Z_cg                  generic_.a_cg_body_v[2]

    VECTOR_3    a_pilot_body_v;
#define A_pilot_body_v          generic_.a_pilot_body_v
#define A_X_pilot               generic_.a_pilot_body_v[0]
#define A_Y_pilot               generic_.a_pilot_body_v[1]
#define A_Z_pilot               generic_.a_pilot_body_v[2]

    VECTOR_3    n_cg_body_v;
#define N_cg_body_v             generic_.n_cg_body_v
#define N_X_cg                  generic_.n_cg_body_v[0]
#define N_Y_cg                  generic_.n_cg_body_v[1]
#define N_Z_cg                  generic_.n_cg_body_v[2]

    VECTOR_3    n_pilot_body_v;
#define N_pilot_body_v          generic_.n_pilot_body_v
#define N_X_pilot               generic_.n_pilot_body_v[0]
#define N_Y_pilot               generic_.n_pilot_body_v[1]
#define N_Z_pilot               generic_.n_pilot_body_v[2]

    VECTOR_3    omega_dot_body_v;
#define Omega_dot_body_v        generic_.omega_dot_body_v
```



```
#define P_dot_body              generic_.omega_dot_body_v[0]
#define Q_dot_body              generic_.omega_dot_body_v[1]
#define R_dot_body              generic_.omega_dot_body_v[2]

/*============================= Velocities ==============================*/

    VECTOR_3    v_local_v;
#define V_local_v               generic_.v_local_v
#define V_north                 generic_.v_local_v[0]
#define V_east                  generic_.v_local_v[1]
#define V_down                  generic_.v_local_v[2]

    VECTOR_3    v_local_rel_ground_v;   /* V rel w.r.t. earth surface   */
#define V_local_rel_ground_v    generic_.v_local_rel_ground_v
#define V_north_rel_ground      generic_.v_local_rel_ground_v[0]
#define V_east_rel_ground       generic_.v_local_rel_ground_v[1]
#define V_down_rel_ground       generic_.v_local_rel_ground_v[2]

    VECTOR_3    v_local_airmass_v;      /* velocity of airmass (steady winds)   */
#define V_local_airmass_v       generic_.v_local_airmass_v
#define V_north_airmass         generic_.v_local_airmass_v[0]
#define V_east_airmass          generic_.v_local_airmass_v[1]
#define V_down_airmass          generic_.v_local_airmass_v[2]

    VECTOR_3    v_local_rel_airmass_v;  /* velocity of veh. relative to airmass */
#define V_local_rel_airmass_v   generic_.v_local_rel_airmass_v
#define V_north_rel_airmass     generic_.v_local_rel_airmass_v[0]
#define V_east_rel_airmass      generic_.v_local_rel_airmass_v[1]
#define V_down_rel_airmass      generic_.v_local_rel_airmass_v[2]

    VECTOR_3    v_local_gust_v; /* linear turbulence components, L frame */
#define V_local_gust_v          generic_.v_local_gust_v
#define U_gust                  generic_.v_local_gust_v[0]
#define V_gust                  generic_.v_local_gust_v[1]
#define W_gust                  generic_.v_local_gust_v[2]

    VECTOR_3    v_wind_body_v;  /* Wind-relative velocities in body axis        */
#define V_wind_body_v           generic_.v_wind_body_v
#define U_body                  generic_.v_wind_body_v[0]
#define V_body                  generic_.v_wind_body_v[1]
#define W_body                  generic_.v_wind_body_v[2]

    DATA    v_rel_wind, v_true_kts, v_rel_ground, v_inertial;
    DATA    v_ground_speed, v_equiv, v_equiv_kts;
    DATA    v_calibrated, v_calibrated_kts;
#define V_rel_wind              generic_.v_rel_wind
#define V_true_kts              generic_.v_true_kts
#define V_rel_ground            generic_.v_rel_ground
#define V_inertial              generic_.v_inertial
#define V_ground_speed          generic_.v_ground_speed
#define V_equiv                 generic_.v_equiv
#define V_equiv_kts             generic_.v_equiv_kts
#define V_calibrated            generic_.v_calibrated
#define V_calibrated_kts        generic_.v_calibrated_kts

    VECTOR_3    omega_body_v;   /* Angular B rates      */
#define Omega_body_v            generic_.omega_body_v
#define P_body                  generic_.omega_body_v[0]
#define Q_body                  generic_.omega_body_v[1]
#define R_body                  generic_.omega_body_v[2]

    VECTOR_3    omega_local_v;  /* Angular L rates      */
#define Omega_local_v           generic_.omega_local_v
#define P_local                 generic_.omega_local_v[0]
```

```
#define Q_local                 generic_.omega_local_v[1]
#define R_local                 generic_.omega_local_v[2]

    VECTOR_3    omega_total_v;  /* Diff btw B & L      */
#define Omega_total_v           generic_.omega_total_v
#define P_total                 generic_.omega_total_v[0]
#define Q_total                 generic_.omega_total_v[1]
#define R_total                 generic_.omega_total_v[2]

    VECTOR_3    euler_rates_v;
#define Euler_rates_v           generic_.euler_rates_v
#define Phi_dot                 generic_.euler_rates_v[0]
#define Theta_dot               generic_.euler_rates_v[1]
#define Psi_dot                 generic_.euler_rates_v[2]

    VECTOR_3    geocentric_rates_v;    /* Geocentric linear velocities */
#define Geocentric_rates_v      generic_.geocentric_rates_v
#define Latitude_dot            generic_.geocentric_rates_v[0]
#define Longitude_dot           generic_.geocentric_rates_v[1]
#define Radius_dot              generic_.geocentric_rates_v[2]

/*============================== Positions ==============================*/

    VECTOR_3    geocentric_position_v;
#define Geocentric_position_v   generic_.geocentric_position_v
#define Lat_geocentric          generic_.geocentric_position_v[0]
#define Lon_geocentric          generic_.geocentric_position_v[1]
#define Radius_to_vehicle       generic_.geocentric_position_v[2]

    VECTOR_3    geodetic_position_v;
#define Geodetic_position_v     generic_.geodetic_position_v
#define Latitude                generic_.geodetic_position_v[0]
#define Longitude               generic_.geodetic_position_v[1]
#define Altitude                generic_.geodetic_position_v[2]

    VECTOR_3    euler_angles_v;
#define Euler_angles_v          generic_.euler_angles_v
#define Phi                     generic_.euler_angles_v[0]
#define Theta                   generic_.euler_angles_v[1]
#define Psi                     generic_.euler_angles_v[2]

/*====================== Miscellaneous quantities ======================*/

    DATA    t_local_to_body_m[3][3];    /* Transformation matrix L to B */
#define T_local_to_body_m       generic_.t_local_to_body_m
#define T_local_to_body_11      generic_.t_local_to_body_m[0][0]
#define T_local_to_body_12      generic_.t_local_to_body_m[0][1]
#define T_local_to_body_13      generic_.t_local_to_body_m[0][2]
#define T_local_to_body_21      generic_.t_local_to_body_m[1][0]
#define T_local_to_body_22      generic_.t_local_to_body_m[1][1]
#define T_local_to_body_23      generic_.t_local_to_body_m[1][2]
#define T_local_to_body_31      generic_.t_local_to_body_m[2][0]
#define T_local_to_body_32      generic_.t_local_to_body_m[2][1]
#define T_local_to_body_33      generic_.t_local_to_body_m[2][2]

    DATA    gravity;            /* Local acceleration due to G  */
#define Gravity                 generic_.gravity

    DATA    centrifugal_relief; /* load factor reduction due to speed */
#define Centrifugal_relief      generic_.centrifugal_relief

    DATA    alpha, beta, alpha_dot, beta_dot;   /* in radians   */
#define Alpha                   generic_.alpha
#define Beta                    generic_.beta
#define Alpha_dot               generic_.alpha_dot
#define Beta_dot                generic_.beta_dot

    DATA    cos_alpha, sin_alpha, cos_beta, sin_beta;
#define Cos_alpha               generic_.cos_alpha
#define Sin_alpha               generic_.sin_alpha
#define Cos_beta                generic_.cos_beta
#define Sin_beta                generic_.sin_beta

    DATA    cos_phi, sin_phi, cos_theta, sin_theta, cos_psi, sin_psi;
#define Cos_phi                 generic_.cos_phi
#define Sin_phi                 generic_.sin_phi
#define Cos_theta               generic_.cos_theta
#define Sin_theta               generic_.sin_theta
#define Cos_psi                 generic_.cos_psi
#define Sin_psi                 generic_.sin_psi

    DATA    gamma_vert_rad, gamma_horiz_rad;    /* Flight path angles   */
#define Gamma_vert_rad          generic_.gamma_vert_rad
#define Gamma_horiz_rad         generic_.gamma_horiz_rad

    DATA    sigma, density, v_sound, mach_number;
#define Sigma                   generic_.sigma
#define Density                 generic_.density
#define V_sound                 generic_.v_sound
#define Mach_number             generic_.mach_number

    DATA    static_pressure, total_pressure, impact_pressure, dynamic_pressure;
#define Static_pressure         generic_.static_pressure
#define Total_pressure          generic_.total_pressure
#define Impact_pressure         generic_.impact_pressure
#define Dynamic_pressure        generic_.dynamic_pressure

    DATA    static_temperature, total_temperature;
#define Static_temperature      generic_.static_temperature
#define Total_temperature       generic_.total_temperature

    DATA    sea_level_radius, earth_position_angle;
#define Sea_level_radius        generic_.sea_level_radius
#define Earth_position_angle    generic_.earth_position_angle

    DATA    runway_altitude, runway_latitude, runway_longitude, runway_heading;
#define Runway_altitude         generic_.runway_altitude
#define Runway_latitude         generic_.runway_latitude
#define Runway_longitude        generic_.runway_longitude
#define Runway_heading          generic_.runway_heading

    DATA    radius_to_rwy;
#define Radius_to_rwy           generic_.radius_to_rwy

    VECTOR_3    d_cg_rwy_local_v;      /* CG rel. to rwy in local coords */
#define D_cg_rwy_local_v        generic_.d_cg_rwy_local_v
#define D_cg_north_of_rwy       generic_.d_cg_rwy_local_v[0]
#define D_cg_east_of_rwy        generic_.d_cg_rwy_local_v[1]
#define D_cg_above_rwy          generic_.d_cg_rwy_local_v[2]

    VECTOR_3    d_cg_rwy_rwy_v; /* CG relative to runway, in rwy coordinates */
#define D_cg_rwy_rwy_v          generic_.d_cg_rwy_rwy_v
#define X_cg_rwy                generic_.d_cg_rwy_rwy_v[0]
#define Y_cg_rwy                generic_.d_cg_rwy_rwy_v[1]
#define H_cg_rwy                generic_.d_cg_rwy_rwy_v[2]

    VECTOR_3    d_pilot_rwy_local_v;    /* pilot rel. to rwy in local coords */
#define D_pilot_rwy_local_v     generic_.d_pilot_rwy_local_v
#define D_pilot_north_of_rwy    generic_.d_pilot_rwy_local_v[0]
#define D_pilot_east_of_rwy     generic_.d_pilot_rwy_local_v[1]
```

```
#define D_pilot_above_rwy       generic_.d_pilot_rwy_local_v[2]

    VECTOR_3   d_pilot_rwy_rwy_v;       /* pilot rel. to rwy, in rwy coords. */
#define D_pilot_rwy_rwy_v       generic_.d_pilot_rwy_rwy_v
#define X_pilot_rwy             generic_.d_pilot_rwy_rwy_v[0]
#define Y_pilot_rwy             generic_.d_pilot_rwy_rwy_v[1]
#define H_pilot_rwy             generic_.d_pilot_rwy_rwy_v[2]


} GENERIC;

extern GENERIC generic_;        /* usually defined in ls_main.c */

/*---------------------------  end of ls_generic.h  ------------------------*/
```

```
/***************************************************************************

        TITLE:          ls_matrix.h

----------------------------------------------------------------------------

        FUNCTION:       Header file for general real matrix routines.

        The routines in this module have come more or less from ref [1].
        Note that, probably due to the heritage of ref [1] (which has a
        FORTRAN version that was probably written first), the use of 1 as
        the first element of an array (or vector) is used. This is accomplished
        in memory by allocating, but not using, the 0 elements in each dimension.
        While this wastes some memory, it allows the routines to be ported more
        easily from FORTRAN (I suspect) as well as adhering to conventional
        matrix notation.  As a result, however, traditional ANSI C convention
        (0-base indexing) is not followed; as the authors of ref [1] point out,
        there is some question of the portability of the resulting routines
        which sometimes access negative indexes. See ref [1] for more details.

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 950222 E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    from Numerical Recipes in C, by Press, et. al.

        CODED BY:       Bruce Jackson

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_matrix.h,v 1.1 1995/02/27 20:02:18 bjax Stab $
$Log: ls_matrix.h,v $
 * Revision 1.1  1995/02/27  20:02:18  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:     [1] Press, William H., et. al, Numerical Recipes in
                            C, 2nd edition, Cambridge University Press, 1992

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define NR_END 1

/* matrix creation & destruction routines */

int *nr_ivector(long nl, long nh);
double **nr_matrix(long nrl, long nrh, long ncl, long nch);

void nr_free_ivector(int *v, long nl, long nh);
void nr_free_matrix(double **m, long nrl, long nrh, long ncl, long nch);


/* Gauss-Jordan inversion routine */

int nr_gaussj(double **a, int n, double **b, int m);

/* Linear equation solution by Gauss-Jordan elimination. a[1..n][1..n] is */
/* the input matrix. b[1..n][1..m] is input containing the m right-hand   */
/* side vectors. On output, a is replaced by its matrix invers, and b is  */
/* replaced by the corresponding set of solution vectors.                 */

/* Note: this routine modified by EBJ to make b optional, if m == 0 */

/* Matrix copy, multiply, and printout routines (by EBJ) */

void nr_copymat(double **orig, int n, double **copy);
void nr_multmat(double **m1, int n, double **m2, double **prod);
void nr_printmat(double **a, int n);
```

```
/***************************************************************************

	TITLE:		ls_sim_control.h

----------------------------------------------------------------------------

	FUNCTION:	LaRCSim simulation control parameters header file

----------------------------------------------------------------------------

	MODULE STATUS:	developmental

----------------------------------------------------------------------------

	GENEALOGY:	Created 18 DEC 1993 by Bruce Jackson

----------------------------------------------------------------------------

	DESIGNED BY:	B. Jackson

	CODED BY:	B. Jackson

	MAINTAINED BY:	guess who

----------------------------------------------------------------------------

	MODIFICATION HISTORY:

	DATE	PURPOSE						BY

	940204	Added "overrun" flag to indicate non-real-time frame.
	940210	Added "vision" flag to indicate use of shared memory.
	940513	Added "max_tape_channels" and "max_time_slices" EBJ
	950308	Increased size of time_stamp and date_string to include
		terminating null char.				EBJ
	950314	Addedf "paused" flag to make this global (was local to
		ls_cockpit routine).				EBJ
	950406	Removed tape_channels parameter, and added end_time, model_hz,
		and term_update_hz parameters.			EBJ

$Header: /aces/larcsim/dev/RCS/ls_sim_control.h,v 1.11 1995/04/07 01:39:09 bjax Exp $
$Log: ls_sim_control.h,v $
 * Revision 1.11  1995/04/07  01:39:09  bjax
 * Removed tape_channels and added end_time, model_hz, and term_update_hz.
 *
 * Revision 1.10  1995/03/15  12:33:29  bjax
 * Added 'paused' flag.
 *
 * Revision 1.9  1995/03/08  12:34:21  bjax
 * Increased size of date_string and time_stamp by 1 to include terminating null;
 * added userid field and include of stdio.h. EBJ
 *
 * Revision 1.8  1994/05/13  20:41:43  bjax
 * Increased size of time_stamp to 8 chars to allow for colons.
 * Added fields "tape_channels" and "time_slices" to allow user to change.
 *
 * Revision 1.7  1994/05/10  15:18:49  bjax
 * Modified write_cmp2 flag to write_ascl flag, since XPLOT 4.00 doesn't
 * support cmp2.  Also added RCS header and log entries in header.
 *

--------------------------------------------------------------------------*/
#include <stdio.h>

#ifndef SIM_CONTROL

typedef struct {

  enum { batch, terminal, GLmouse, cockpit } sim_type;
  char simname[64];	/* name of simulation */
  int run_number;	/* run number of this session			*/
  char date_string[7];	/* like "931220" */
  char time_stamp[9];	/* like "13:00:00" */
  char userid[L_cuserid]; /* who is running this sim */
  long time_slices;	/* number of points that can be recorded (circ buff) */
  int write_av;		/* will be writing out an Agile_VU file after run */
  int write_mat;	/* will be writing out a matrix script of session */
  int write_tab;	/* will be writing out a tab-delimited time history */
  int write_ascl;	/* will be writing out a GetData ASCII 1 file */
  int save_spacing;	/* spacing between data points when recording
			   data to memory; 0 = every point, 1 = every
			   other point; 2 = every fourth point, etc. */
  int write_spacing;	/* spacing between data points when writing
			   output files; 0 = every point, 1 = every
			   other point; 2 = every fourth point, etc. */
  int overrun;		/* indicates, if non-zero, a frame overrun
			   occurred in the previous frame. Suitable for
			   setting a display flag or writing an error
			   message.					*/
  int vision;		/* indicates, if non-zero, marriage to LaRC VISION
			   software (developed A. Dare and J. Burley of the
			   former Cockpit Technologies Branch) */
  int debug;		/* indicates, if non-zero, to operate in debug mode
			   which implies disable double-buffering and synch.
			   attempts to avoid errors */
  int paused;		/* indicates simulation is paused */
  float end_time;	/* end of simulation run value */
  float model_hz;	/* current inner loop frame rate */
  float term_update_hz; /* current terminal refresh frequency */

} SIM_CONTROL;

extern SIM_CONTROL sim_control_;

#endif

/*------------------------  end of ls_sim_control.h  ----------------------*/
```

```
/***************************************************************************

        TITLE:          ls_sym.h

----------------------------------------------------------------------------

        FUNCTION:       Header file for symbol table routines

----------------------------------------------------------------------------

        MODULE STATUS:  production

----------------------------------------------------------------------------

        GENEALOGY:      Created 930629 by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       same

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950227  Added header and declarations for ls_print_findsym_error(),
                ls_get_double(), and ls_get_double() routines.  EBJ

        950302  Added structure for symbol description.         EBJ

        950306  Added ls_get_sym_val() and ls_set_sym_val() routines.
                This is now the production version.             EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_sym.h,v 1.9 1995/03/07 12:52:33 bjax Stab $
$Log: ls_sym.h,v $
 * Revision 1.9  1995/03/07  12:52:33  bjax
 * This production version now specified ls_get_sym_val() and ls_put_sym_val().
 *
 * Revision 1.6.1.2  1995/03/06  18:45:41  bjax
 * Added def'n of ls_get_sym_val() and ls_set_sym_val(); changed symbol_rec
 * Addr field from void * to char *.
 *  EBJ
 *
 * Revision 1.6.1.1  1995/03/03  01:17:44  bjax
 * Experimental version with just ls_get_double and ls_set_double() routines.
 *
 * Revision 1.6  1995/02/27  19:50:49  bjax
 * Added header and declarations for ls_print_findsym_error(),
 * ls_get_double(), and ls_set_double().  EBJ
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:
```



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

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

/* Return codes */

#define SYM_NOT_LOADED -2
#define SYM_UNEXPECTED_ERR -1
#define SYM_OK 0
#define SYM_OPEN_ERR 1
#define SYM_NO_SYMS 2
#define SYM_MOD_NOT_FOUND 3
#define SYM_VAR_NOT_FOUND 4
#define SYM_NOT_SCALAR 5
#define SYM_NOT_STATIC 6
#define SYM_MEMORY_ERR 7
#define SYM_UNMATCHED_PAREN 8
#define SYM_BAD_SYNTAX 9
#define SYM_INDEX_BOUNDS_ERR 10

typedef enum {Unknown, Char, UChar, SHint, USHint, Sint, Uint, Slng, Ulng, flt, dbl}
 vartype;

typedef char            SYMBOL_NAME[64];
typedef vartype         SYMBOL_TYPE;


typedef struct
{
    SYMBOL_NAME Mod_Name;
    SYMBOL_NAME Par_Name;
    SYMBOL_TYPE Par_Type;
    SYMBOL_NAME Alias;
    char        *Addr;
}       symbol_rec;


extern int      ls_findsym( const char *modname, const char *varname,
                            char **addr, vartype *vtype );

extern void     ls_print_findsym_error(int result,
                                       char *mod_name,
                                       char *var_name);

extern double   ls_get_double(vartype sym_type, void *addr );

extern void     ls_set_double(vartype sym_type, void *addr, double value );

extern double   ls_get_sym_val( symbol_rec *symrec, int *error );

        /* This routine attempts to return the present value of the symbol
           described in symbol_rec. If Addr is non-zero, the value of that
           location, interpreted as type double, will be returned. If Addr
```

```
          is zero, and Mod_Name and Par_Name are both not null strings,
          the ls_findsym() routine is used to try to obtain the address
          by looking at debugger symbol tables in the executable image, and
          the value of the double contained at that address is returned,
          and the symbol record is updated to contain the address of that
          symbol. If an error is discovered, 'error' will be non-zero and
          and error message is printed on stderr.                  */

extern void     ls_set_sym_val( symbol_rec *symrec, double value );

        /* This routine sets the value of a double at the location pointed
          to by the symbol_rec's Addr field, if Addr is non-zero. If Addr
          is zero, and Mod_Name and Par_Name are both not null strings,
          the ls_findsym() routine is used to try to obtain the address
          by looking at debugger symbol tables in the executable image, and
          the value of the double contained at that address is returned,
          and the symbol record is updated to contain the address of that
          symbol. If an error is discovered, 'error' will be non-zero and
          and error message is printed on stderr.                  */
```

```
/***************************************************************************

        TITLE:  ls_tape.h

----------------------------------------------------------------------------

        FUNCTION:       Tape structure header file

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      920806 version for IRIS from Mac file

----------------------------------------------------------------------------

        DESIGNED BY:    B Jackson

        CODED BY:       B Jackson

        MAINTAINED BY:  B Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        931008  Added Max and Min value recording to save later sweeps of
                data for min and max...                         EBJ

        950302  Moved symbol information structure to ls_sym.h  EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_tape.h,v 1.6 1995/04/07 01:43:08 bjax Exp $
$Log: ls_tape.h,v $
 * Revision 1.6  1995/04/07  01:43:08  bjax
 * Added #ifndef/#endif pair to protect from multiple invocations;
 * changed DATA types to SCALAR types; added Length field.
 *
 * Revision 1.5  1995/03/03  02:03:27  bjax
 * Moved def's of SYMBOL_NAME and SYMBOL_TYPE to ls_sym.h;
 * modified the CHANNEL structure to use new symbol_rec type
 * defined in ls_sym.h. EBJ
 *
 * Revision 1.4  1994/05/13  20:43:11  bjax
 * Changed number of MAX_SLICES from 10K to 32K.
 *
 * Revision 1.3  1993/12/20  16:49:04  bjax
 * Cleaned up the time slice vector; now a simple pointer to a DATA type;
 * application can treat it like the array it is.  EBJ
 *
 * Revision 1.2  1993/10/08  22:05:33  bjax
 * Added standard header; added min-max value element to each channel.  EBJ
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
#include "ls_sym.h"
#ifndef TAPE
#define MAX_TAPE_CHANNELS 1024

typedef struct
{
        symbol_rec      Symbol;
        SCALAR          Max_value, Min_value;
        SCALAR          *Data;
}       CHANNEL;


typedef struct
{
        int             Num_Chan;
        long            Length;
        int             First, Current, Next, Last;
        SCALAR          T_First, T_Current, T_Next, T_Last;
        SCALAR          Factor_1, Factor_2;
        SCALAR          *T_Stamp;               /* Pointer to an array of time stamp
s */
        CHANNEL         *Chan[ MAX_TAPE_CHANNELS ];     /* An array of handles
                                                                to Chan structures *
/
}       TAPE;

#endif
```

```
/***************************************************************************

        TITLE:  ls_types.h

----------------------------------------------------------------------------

        FUNCTION:       LaRCSim type definitions header file

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 15 DEC 1993 by Bruce Jackson from old
                        ls_eom.h header

----------------------------------------------------------------------------

        DESIGNED BY:    B. Jackson

        CODED BY:       B. Jackson

        MAINTAINED BY:  guess who

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

--------------------------------------------------------------------------*/
/* SCALAR type is used throughout equations of motion code - sets precision */

typedef double SCALAR;

typedef SCALAR VECTOR_3[3];

/* DATA type is old style; this statement for continuity */

#define DATA SCALAR

/* -------------------------- end of ls_types.h ------------------------*/
```

```
/***************************************************************************

        TITLE:          LaRCsim.c

----------------------------------------------------------------------------

        FUNCTION:       Top level routine for LaRCSIM.  Includes
                        global variable declarations.

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      Written 921230 by Bruce Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    EBJ

        CODED BY:       EBJ

        MAINTAINED BY:  EBJ

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        930111  Added "progname" variable to keep name of invoking command.
                                                                        EBJ
        931012  Removed altitude < 0. test to support gear development. EBJ
        931214  Added various pressures (Impact, Dynamic, Static, etc.) EBJ
        931215  Adopted new generic variable structure.                 EBJ
        931218  Added command line options decoding.                    EBJ
        940110  Changed file type of matrix file to ".m"                EBJ
        940513  Renamed this routine "LaRCsim.c" from "ls_main.c"       EBJ
        940513  Added time_stamp routine,  t_stamp.                     EBJ
        950225  Added options flag, 'i', to set I/O output rate.        EBJ
        950306  Added calls to ls_get_settings() and ls_put_settings()  EBJ
        950314  Options flag 'i' now reads IC file; 'o' is output rate  EBJ
        950406  Many changes: added definition of default value macros;
                removed local variables term_update_hz, model_dt, endtime,
                substituted sim_control_ globals for these; removed
                initialization of sim_control_.tape_channels; moved optarg
                to generic extern; added mod_end_time & mod_buf_size flags
                and temporary buffer_time and data_rate locals to
                ls_checkopts(); added additional command line switches '-s'
                and '-b'; made psuedo-mandatory file names for data output
                switches; considerable rewrite of logic for setting data
                buffer length and interleave parameters; updated '-h' help
                output message; added protection logic to calculations of
                these parameters; added check of return value on first call
                to ls_cockpit() so <esc> abort works from initial pause
                state; added call to ls_unsync() immediately following
                first ls_sync() call, if paused (to avoid alarm clock
                timeout); moved call to ls_record() into non-paused
                multiloop path (was filling buffer with identical data
                during pause); put check of paused flag before calling sync
                routine ls_pause(); and added call to exit() on termination.

$Header: /aces/larcsim/dev/RCS/LaRCsim.c,v 1.4.1.7 1995/04/07 01:04:37 bjax Exp $

$Log: LaRCsim.c,v $
 * Revision 1.4.1.7  1995/04/07  01:04:37  bjax
 * Many changes made to support storage of sim options from run to run,
 * as well as restructuring storage buffer sizing and some loop logic
 * changes. See the modification log for details.
 *
 * Revision 1.4.1.6  1995/03/29  16:12:09  bjax
 * Added argument to -o switch; changed run loop to pass dt=0
 * if in paused mode. EBj
 *
 * Revision 1.4.1.5  1995/03/15  12:30:20  bjax
 * Set paused flag to non-zero by default; moved 'i' I/O rate flag
 * switch to 'o'; made 'i' an initial conditions file switch; added
 * null string to ls_get_settings() call so that default settings
 * file will be read. EBJ
 *
 * Revision 1.4.1.4  1995/03/08  12:31:34  bjax
 * Added userid retrieval and proper termination of time & date strings.
 *
 * Revision 1.4.1.3  1995/03/08  12:00:21  bjax
 * Moved setting of default options to ls_setdefopts from
 * ls_checkopts; rearranged order of ls_get_settings() call
 * to between ls_setdefopts and ls_checkopts, so command
 * line options will override settings file options.
 * EBJ
 *
 * Revision 1.4.1.2  1995/03/06  18:48:49  bjax
 * Added calles to ls_get_settings() and ls_put_settings(); added
 * passing of dt and init flags in ls_model(). EBJ
 *
 * Revision 1.4.1.1  1995/03/03  02:23:08  bjax
 * Beta version for LaRCsim, version 1.4
 *
 * Revision 1.3.2.7  1995/02/27  20:00:21  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.2.6  1995/02/25  16:52:31  bjax
 * Added 'i' option to set I/O iteration rate. EBJ
 *
 * Revision 1.3.2.5  1995/02/06  19:33:15  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.2.4  1995/02/06  19:30:30  bjax
 * Oops, should really compile these before checking in. Fixed capitailzation of
 * Initialize in ls_loop parameter.
 *
 * Revision 1.3.2.3  1995/02/06  19:25:44  bjax
 * Moved main simulation loop into subroutine ls_loop. EBJ
 *
 * Revision 1.3.2.2  1994/05/20  21:46:45  bjax
 * A little better logic on checking for option arguments.
 *
 * Revision 1.3.2.1  1994/05/20  19:29:51  bjax
 * Added options arguments to command line.
 *
 * Revision 1.3.1.16  1994/05/17  15:08:45  bjax
 * Corrected so that full name to directyr and file is saved
 * in new global variable "fullname"; this allows symbol table
 * to be extracted when in another default directory.
 *
 * Revision 1.3.1.15  1994/05/17  14:50:24  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.14  1994/05/17  14:50:23  bjax
 * Rebuilt LaRCsim
```

```
 *
 * Revision 1.3.1.13  1994/05/17  14:50:21  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.12  1994/05/17  14:50:20  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.11  1994/05/17  13:56:24  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.10  1994/05/17  13:23:03  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.9  1994/05/17  13:20:03  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.8  1994/05/17  13:19:23  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.7  1994/05/17  13:18:29  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.6  1994/05/17  13:16:30  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.5  1994/05/17  13:03:44  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.4  1994/05/17  13:03:38  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.3  1994/05/17  12:49:08  bjax
 * Rebuilt LaRCsim
 *
 * Revision 1.3.1.2  1994/05/17  12:48:45  bjax
 * *** empty log message ***
 *
 * Revision 1.3.1.1  1994/05/13  20:39:17  bjax
 * Top of 1.3 branch.
 *
 * Revision 1.2  1994/05/13  19:51:50  bjax
 * Skip rev
 *

--------------------------------------------------------------------------

        REFERENCES:

--------------------------------------------------------------------------

        CALLED BY:

--------------------------------------------------------------------------

        CALLS TO:

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
```



```
#include "ls_types.h"
#include "ls_constants.h"
#include "ls_generic.h"
#include "ls_sim_control.h"
#include "ls_tape.h"
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

/* global variable declarations */

TAPE            *Tape;
GENERIC         generic_;
SIM_CONTROL     sim_control_;

SCALAR          Simtime;

#define DEFAULT_TERM_UPDATE_HZ 20
#define DEFAULT_MODEL_HZ 120
#define DEFAULT_END_TIME 3600.
#define DEFAULT_SAVE_SPACING 8
#define DEFAULT_WRITE_SPACING 1
#define MAX_FILE_NAME_LENGTH 80

/* global variables */

char    *progname;
char    *fullname;

/* file variables - default simulation settings */

static float io_dt;
static float speedup;
static char  asclname[MAX_FILE_NAME_LENGTH] = "run.asc1";
static char  tabname[MAX_FILE_NAME_LENGTH]  = "run.dat";
static char  fltname[MAX_FILE_NAME_LENGTH]  = "run.flt";
static char  matname[MAX_FILE_NAME_LENGTH]  = "run.m";


void ls_stamp()
{
    char rcsid[] = "$Id: LaRCsim.c,v 1.4.1.7 1995/04/07 01:04:37 bjax Exp $";
    char revid[] = "$Revision: 1.4.1.7 $";
    char dateid[] = "$Date: 1995/04/07 01:04:37 $";
    struct tm *nowtime;
    time_t nowtime_t;
    long date;

    printf("\nLaRCsim %s, %s\n\n", revid, dateid);          /* report version of LaR
Csim */

    nowtime_t = time( 0 );
    nowtime = localtime( &nowtime_t ); /* set fields to correct time values */
    date = (nowtime->tm_year)*10000
         + (nowtime->tm_mon + 1)*100
         + (nowtime->tm_mday);
    sprintf(sim_control_.date_string, "%06d\0", date);
    sprintf(sim_control_.time_stamp, "%02d:%02d:%02d\0",
        nowtime->tm_hour, nowtime->tm_min, nowtime->tm_sec);
```

```
    cuserid( sim_control_.userid );     /* set up user id */

    return;
}

void ls_setdefopts()
{
    /* set default values for most options */

    sim_control_.debug = 0;             /* change to non-zero if in dbx! */
    sim_control_.vision = 0;
    sim_control_.write_av = 0;          /* write Agile-Vu '.flt' file */
    sim_control_.write_mat = 0;         /* write matrix-x/matlab script */
    sim_control_.write_tab = 0;         /* write tab delim. history file */
    sim_control_.write_ascl = 0;        /* write GetData file */
    sim_control_.sim_type = cockpit;    /* hook up to cockpit */
    sim_control_.save_spacing = DEFAULT_SAVE_SPACING;
                                        /* interpolation on recording */
    sim_control_.write_spacing = DEFAULT_WRITE_SPACING;
                                        /* interpolation on output */
    sim_control_.end_time = DEFAULT_END_TIME;
    sim_control_.model_hz = DEFAULT_MODEL_HZ;
    sim_control_.term_update_hz = DEFAULT_TERM_UPDATE_HZ;
    sim_control_.time_slices = DEFAULT_END_TIME*DEFAULT_MODEL_HZ/DEFAULT_SAVE_SPACING;
    sim_control_.paused = 1;

    speedup = 1.0;
}

/* return result codes from ls_checkopts */

#define OPT_OK 0
#define OPT_ERR 1

extern char *optarg;
extern int optind;

int ls_checkopts(argc, argv)    /* check and set options flags */
  int argc;
  char *argv[];
  {
    int c;
    int opt_err = 0;
    int mod_end_time = 0;
    int mod_buf_size = 0;
    float buffer_time, data_rate;

    /* set default values */

    buffer_time = sim_control_.time_slices * sim_control_.save_spacing / sim_control_.mo
del_hz;
    data_rate   = sim_control_.model_hz / sim_control_.save_spacing;

    while ((c = getopt(argc, argv, "Aa:b:de:f:hi:kmo:r:s:t:x:")) != EOF)
        switch (c) {
            case 'A':
                if (sim_control_.sim_type == GLmouse)
                  {
                    fprintf(stderr, "Cannot specify both keyboard (k) and ACES (A) cockp
its option\n");
                    fprintf(stderr, "Keyboard operation assumed.\n");
                    break;
                  }
                sim_control_.sim_type = cockpit;
                break;
            case 'a':
                sim_control_.write_av = 1;
                if (optarg != NULL)
                if (*optarg != '-')
                    strncpy(fltname, optarg, MAX_FILE_NAME_LENGTH);
                else
                    optind--;
                break;
            case 'b':
                buffer_time = atof(optarg);
                if (buffer_time <= 0.) opt_err = -1;
                mod_buf_size++;
                break;
            case 'd':
                sim_control_.debug = 1;
                break;
            case 'e':
                sim_control_.end_time = atof(optarg);
                mod_end_time++;
                break;
            case 'f':
                sim_control_.model_hz = atof(optarg);
                break;
            case 'h':
                opt_err = 1;
                break;
            case 'i':
                ls_get_settings( optarg );
                break;
            case 'k':
                sim_control_.sim_type = GLmouse;
                break;
            case 'm':
                sim_control_.vision = 1;
                break;
            case 'o':
                sim_control_.term_update_hz = atof(optarg);
                if (sim_control_.term_update_hz <= 0.) opt_err = 1;
                break;
            case 'r':
                sim_control_.write_mat = 1;
                if (optarg != NULL)
                if (*optarg != '-')
                    strncpy(matname, optarg, MAX_FILE_NAME_LENGTH);
                else
                    optind--;
                break;
            case 's':
                data_rate = atof(optarg);
                if (data_rate <= 0.) opt_err = -1;
                break;
            case 't':
                sim_control_.write_tab = 1;
                if (optarg != NULL)
                if (*optarg != '-')
                    strncpy(tabname, optarg, MAX_FILE_NAME_LENGTH);
                else
                    optind--;
                break;
            case 'x':
                sim_control_.write_ascl = 1;
                if (optarg != NULL)
                if (*optarg != '-')
                    strncpy(asclname, optarg, MAX_FILE_NAME_LENGTH);
```

```
            else
                optind--;
            break;
        default:
            opt_err = 1;

        }

    if (opt_err)
      {
        fprintf(stderr, "Usage: %s [-options]\n", progname);
        fprintf(stderr, "\n");
        fprintf(stderr, "  where [-options] is zero or more of the following:\n");
        fprintf(stderr, "\n");
        fprintf(stderr, "  [A|k]              Run mode: [A]CES cockpit    [default]\n");
        fprintf(stderr, "                               or [k]eyboard\n");
        fprintf(stderr, "\n");
        fprintf(stderr, "  [i <filename>]    [i]nitial conditions filename\n");
        fprintf(stderr, "\n");
        fprintf(stderr, "  [f <value>]       Iteration rate [f]requency, Hz (default is %5
.2f Hz)\n",
                                 sim_control_.model_hz);
        fprintf(stderr, "\n");
        fprintf(stderr, "  [o <value>]       Display [o]utput frequency, Hz (default is %5
.2f Hz)\n",
                                 sim_control_.term_update_hz);
        fprintf(stderr, "\n");
        fprintf(stderr, "  [s <value>]       Data storage frequency, Hz (default is %5.2f
Hz)\n",
                                 data_rate);
        fprintf(stderr, "\n");
        fprintf(stderr, "  [e <value>]       [e]nd time in seconds (default %5.1f seconds)
\n",
                                 sim_control_.end_time);
        fprintf(stderr, "\n");
        fprintf(stderr, "  [b <value>]       circular time history storage [b]uffer size,
in seconds \n");
        fprintf(stderr, "                    (default %5.1f seconds) (normally same as end
 time)\n",
                                 sim_control_.time_slices*sim_control_.sa
ve_spacing/
                                         sim_control_.model_hz);
        fprintf(stderr, "\n");
        fprintf(stderr, "  [atxr [<filename>]] Output: [a]gile-vu  (default name: %s )\n
", fltname);
        fprintf(stderr, "                          and/or [t]ab delimited ( '' name: %s )\n
", tabname);
        fprintf(stderr, "                          and/or [x]plot     (default name: %s)\n"
, asclname);
        fprintf(stderr, "                          and/or mat[r]ix script ( '' name: %s    )
\n", matname);
        fprintf(stderr, "\n");
        return OPT_ERR;
      }

/* calculate additional controls */

    io_dt = 1.0/sim_control_.term_update_hz;

    sim_control_.save_spacing = (int) (0.5 + sim_control_.model_hz / data_rate);
    if (sim_control_.save_spacing < 1) sim_control_.save_spacing = 1;

    sim_control_.time_slices = buffer_time * sim_control_.model_hz / sim_control_.save_s
pacing;
    if (sim_control_.time_slices < 2) sim_control_.time_slices = 2;

    return OPT_OK;
  }


void ls_loop( dt, initialize )

SCALAR dt;
int initialize;

{
    ls_step( dt, initialize );
    if (sim_control_.sim_type == cockpit ) ls_ACES();
    ls_aux();
    ls_model( dt, initialize );
    ls_accel();
}



main(argc, argv)
int argc;
char *argv[];
{
        int     i, multiloop, abrt;
        SCALAR  model_dt;

        fullname = argv[0];               /* point to full directory & path name of ou
r program */
        progname = basename(argv[0]);     /* point to name of program that invoked us
 */
        strcpy(sim_control_.simname, progname); /* really should check for overflow*
/

        ls_setdefopts();                  /* set default options */

        ls_get_settings( "" );            /* let settings file override them */

        if (ls_checkopts(argc, argv) == OPT_ERR) return 1;      /* and then command
line opts */

        ls_stamp();   /* ID stamp; record time and date of run */

        if (speedup == 0.0)
          {
            fprintf(stderr, "%s: Cannot run with speedup of 0.\n", progname);
            return 1;
          }

        model_dt = 1./sim_control_.model_hz;

        if (io_dt < model_dt)
        {
            fprintf(stderr, "%s: Cannot run I/O faster than model.\n", progname);
            return 1;
        }

        multiloop = (int) (io_dt/model_dt/fabs(speedup));
        model_dt = io_dt/multiloop;


        ls_init();

        ls_record();
        if (speedup > 0)
```

```
        {
            abrt = ls_cockpit();
            if (abrt)
                {
                    ls_cockpit_exit();
                    exit( EXIT_SUCCESS );
                }
            ls_sync(io_dt);
            if (sim_control_.paused) ls_unsync();
        }

        do
          {
            for(i=0;i<multiloop;i++)
              {
                if (sim_control_.paused)
                    ls_loop( 0.0, 0);
                else
                    {
                        ls_loop( model_dt, 0);
                        ls_record();
                    }
              }
            if (speedup > 0)
              {
                abrt = ls_cockpit();
                if (!sim_control_.paused) ls_pause();
                if (abrt) Simtime = sim_control_.end_time+model_dt;
              }
          }
        while (Simtime < sim_control_.end_time);

        if (speedup > 0)
          {
            ls_unsync();
            ls_cockpit_exit();
          }

        if (sim_control_.write_mat ) ls_writemat(  matname  );
        if (sim_control_.write_av  ) ls_writeav(   fltname  );
        if (sim_control_.write_tab ) ls_writetab(  tabname  );
        if (sim_control_.write_asc1) ls_writeasc1( asc1name );

        ls_put_settings();

        exit( EXIT_SUCCESS );
}

/*
Mon Feb  6 14:33:15 EST 1995
bjax
*/
/*
Mon Feb 27 15:00:20 EST 1995
bjax
*/
```

```
/***************************************************************************

    TITLE:          atmos_62

----------------------------------------------------------------------------

    FUNCTION:       1962 atmosphere table interpolation routine

----------------------------------------------------------------------------

    MODULE STATUS:  developmental

----------------------------------------------------------------------------

    GENEALOGY:  Created 920827 by Bruce Jackson as part of the C-castle
                development effort.

----------------------------------------------------------------------------

    DESIGNED BY:    B. Jackson

    CODED BY:       B. Jackson

    MAINTAINED BY:  B. Jackson

----------------------------------------------------------------------------

    MODIFICATION HISTORY:

        DATE    PURPOSE                                              BY
        931220  Added ambient pressure and temperature as outputs.   EBJ
        940111  Changed includes from "ls_eom.h" to "ls_types.h" and
                "ls_constants.h"; changed DATA to SCALAR types.      EBJ

----------------------------------------------------------------------------

    REFERENCES:

    [ 1]        Hornbeck, Robert W.: "Numerical Methods", Prentice-Hall,
                1975.  ISBN  0-13-626614-2

----------------------------------------------------------------------------

    CALLED BY:

----------------------------------------------------------------------------

    CALLS TO:

----------------------------------------------------------------------------

    INPUTS:

----------------------------------------------------------------------------

    OUTPUTS:

--------------------------------------------------------------------------*/

#include "ls_types.h"
#include "ls_constants.h"
#include <math.h>

#define alt_0   d_a_table[index  ][0]
#define alt_1   d_a_table[index+1][0]
#define den_0   d_a_table[index  ][1]
#define den_1   d_a_table[index+1][1]
#define sps_0   d_a_table[index  ][2]
#define sps_1   d_a_table[index+1][2]
#define gden_0  d_a_table[index  ][3]
#define gden_1  d_a_table[index+1][3]
#define gsps_0  d_a_table[index  ][4]
#define gsps_1  d_a_table[index+1][4]

#define MAX_ALT_INDEX 121
#define DELT_ALT 2000.
#define HLEV 36089.
#define TAMBO 518.7
#define PAMBO 2113.8
#define MAX_ALTITUDE 240000.

void ls_atmos( SCALAR altitude, SCALAR * sigma, SCALAR * v_sound,
               SCALAR * t_amb, SCALAR * p_amb )
{

    int         index;
    SCALAR      daltp, daltn, daltp3, daltn3, density;
    SCALAR      t_amb_r, p_amb_r;
    static SCALAR       d_a_table[MAX_ALT_INDEX][5] =
    {
        (     0.,      2.37701E-03,    1.11642E+03,    0.00000E+00,    0.00000E+00
),
        (  2000.,      2.24098E-03,    1.10872E+03,    1.92857E-12,   -1.75815E-08
),
        (  4000.,      2.11099E-03,    1.10097E+03,    1.34570E-12,   -1.21740E-08
),
        (  6000.,      1.98684E-03,    1.09315E+03,    1.44862E-12,   -1.47225E-08
),
        (  8000.,      1.86836E-03,    1.08529E+03,    1.36481E-12,   -1.44359E-08
),
        ( 10000.,      1.75537E-03,    1.07736E+03,    1.32716E-12,   -1.45340E-08
),
        ( 12000.,      1.64768E-03,    1.06938E+03,    1.27657E-12,   -1.44280E-08
),
        ( 14000.,      1.54511E-03,    1.06133E+03,    1.24656E-12,   -1.62540E-08
),
        ( 16000.,      1.44751E-03,    1.05323E+03,    1.19220E-12,   -1.50560E-08
),
        ( 18000.,      1.35469E-03,    1.04506E+03,    1.15463E-12,   -1.65220E-08
),
        ( 20000.,      1.26649E-03,    1.03683E+03,    1.11926E-12,   -1.63562E-08
),
        ( 22000.,      1.18276E-03,    1.02853E+03,    1.07333E-12,   -1.70533E-08
),
        ( 24000.,      1.10333E-03,    1.02016E+03,    1.03743E-12,   -1.59305E-08
),
        ( 26000.,      1.02805E-03,    1.01173E+03,    1.00195E-12,   -2.27248E-08
),
        ( 28000.,      9.56760E-04,    1.00322E+03,    9.39764E-13,    3.29851E-10
),
        ( 30000.,      8.89320E-04,    9.94641E+02,    1.01399E-12,   -8.80946E-08
),
        ( 32000.,      8.25570E-04,    9.85988E+02,    5.39268E-13,    2.41048E-07
),
        ( 34000.,      7.65380E-04,    9.77258E+02,    2.16894E-12,   -9.91599E-07
),
        ( 36000.,      7.08600E-04,    9.68448E+02,   -4.10001E-12,    3.60535E-06
),
        ( 38000.,      6.44190E-04,    9.68053E+02,    2.78612E-12,   -8.07290E-07
),
```

```
    {  40000.,   5.85146E-04,   9.68053E+02,   1.00455E-12,   2.16313E-07
},
    {  42000.,   5.31517E-04,   9.68053E+02,   1.31819E-12,  -5.79609E-08
},
    {  44000.,   4.82801E-04,   9.68053E+02,   1.09217E-12,   1.55309E-08
},
    {  46000.,   4.38554E-04,   9.68053E+02,   1.01661E-12,  -4.16262E-09
},
    {  48000.,   3.98359E-04,   9.68053E+02,   9.19375E-13,   1.11961E-09
},
    {  50000.,   3.61850E-04,   9.68053E+02,   8.34886E-13,  -3.15801E-10
},
    {  52000.,   3.28686E-04,   9.68053E+02,   7.58579E-13,   1.43600E-10
},
    {  54000.,   2.98561E-04,   9.68053E+02,   6.89297E-13,  -2.58597E-10
},
    {  56000.,   2.71197E-04,   9.68053E+02,   6.25735E-13,   8.90788E-10
},
    {  58000.,   2.46341E-04,   9.68053E+02,   5.69765E-13,  -3.30456E-09
},
    {  60000.,   2.23765E-04,   9.68053E+02,   5.15206E-13,   1.23274E-08
},
    {  62000.,   2.03256E-04,   9.68053E+02,   4.69912E-13,  -4.60052E-08
},
    {  64000.,   1.84627E-04,   9.68053E+02,   4.25146E-13,   1.71693E-07
},
    {  66000.,   1.67616E-04,   9.68314E+02,   2.56502E-13,  -2.49268E-07
},
    {  68000.,   1.51855E-04,   9.68676E+02,   4.23844E-13,   9.76878E-07
},
    {  70000.,   1.37615E-04,   9.71034E+02,   3.29621E-13,  -6.64245E-07
},
    {  72000.,   1.24744E-04,   9.72390E+02,   3.11170E-13,   1.77102E-07
},
    {  74000.,   1.13107E-04,   9.73745E+02,   2.76697E-13,  -4.56627E-08
},
    {  76000.,   1.02584E-04,   9.75099E+02,   2.53043E-13,   4.04902E-09
},
    {  78000.,   9.30660E-05,   9.76450E+02,   2.18633E-13,   2.49667E-08
},
    {  80000.,   8.44530E-05,   9.77799E+02,   2.29927E-13,  -1.06916E-07
},
    {  82000.,   7.67140E-05,   9.78950E+02,   1.72660E-13,   1.05696E-07
},
    {  84000.,   6.97010E-05,   9.80290E+02,   1.68432E-13,  -3.23682E-08
},
    {  86000.,   6.33490E-05,   9.81620E+02,   1.45113E-13,   8.77690E-09
},
    {  88000.,   5.75880E-05,   9.82950E+02,   1.37617E-13,  -2.73938E-09
},
    {  90000.,   5.23700E-05,   9.84280E+02,   1.18918E-13,   2.18061E-09
},
    {  92000.,   4.76350E-05,   9.85610E+02,   1.11210E-13,  -5.98306E-09
},
    {  94000.,   4.33410E-05,   9.86930E+02,   9.77408E-14,   6.75162E-09
},
    {  96000.,   3.94430E-05,   9.88260E+02,   9.18264E-14,  -6.02343E-09
},
    {  98000.,   3.59080E-05,   9.89580E+02,   7.94534E-14,   2.34210E-09
},
    { 100000.,   3.26960E-05,   9.90900E+02,   7.48600E-14,  -3.34498E-09
},
    { 102000.,   2.97810E-05,   9.92210E+02,   6.66067E-14,  -3.96219E-09
},
    { 104000.,   2.71320E-05,   9.93530E+02,   5.77131E-14,   3.41937E-08
},
    { 106000.,   2.46980E-05,   9.95410E+02,   2.50410E-14,   7.07187E-07
},
    { 108000.,   2.24140E-05,   9.99070E+02,   6.71229E-14,  -1.92943E-07
},
    { 110000.,   2.03570E-05,   1.00272E+03,   4.69675E-14,   4.95832E-08
},
    { 112000.,   1.85010E-05,   1.00636E+03,   4.65069E-14,  -2.03903E-08
},
    { 114000.,   1.68270E-05,   1.00998E+03,   4.00047E-14,   1.97789E-09
},
    { 116000.,   1.53150E-05,   1.01359E+03,   3.64744E-14,  -2.52130E-09
},
    { 118000.,   1.39480E-05,   1.01719E+03,   3.15976E-14,  -6.89271E-09
},
    { 120000.,   1.27100E-05,   1.02077E+03,   3.06351E-14,   9.21465E-11
},
    { 122000.,   1.15920E-05,   1.02434E+03,   2.58618E-14,  -8.47587E-09
},
    { 124000.,   1.05790E-05,   1.02789E+03,   2.34176E-14,   3.81135E-09
},
    { 126000.,   9.66010E-06,   1.03144E+03,   2.16178E-14,  -6.76951E-09
},
    { 128000.,   8.82710E-06,   1.03497E+03,   1.89611E-14,  -6.73330E-09
},
    { 130000.,   8.07070E-06,   1.03848E+03,   1.74377E-14,   3.70270E-09
},
    { 132000.,   7.38380E-06,   1.04199E+03,   1.55382E-14,  -8.07752E-09
},
    { 134000.,   6.75940E-06,   1.04548E+03,   1.41595E-14,  -1.39263E-09
},
    { 136000.,   6.19160E-06,   1.04896E+03,   1.27239E-14,  -1.35196E-09
},
    { 138000.,   5.67490E-06,   1.05243E+03,   1.15951E-14,  -8.19953E-09
},
    { 140000.,   5.20450E-06,   1.05588E+03,   1.03459E-14,   4.15010E-09
},
    { 142000.,   4.77570E-06,   1.05933E+03,   9.42149E-15,  -8.40086E-09
},
    { 144000.,   4.38470E-06,   1.06276E+03,   8.66820E-15,  -5.46671E-10
},
    { 146000.,   4.02820E-06,   1.06618E+03,   7.65573E-15,  -4.41246E-09
},
    { 148000.,   3.70260E-06,   1.06959E+03,   7.05890E-15,   3.19650E-09
},
    { 150000.,   3.40520E-06,   1.07299E+03,   6.40867E-15,  -2.33736E-08
},
    { 152000.,   3.13330E-06,   1.07637E+03,   5.55641E-15,   6.02977E-08
},
    { 154000.,   2.88480E-06,   1.07975E+03,   6.46568E-15,  -2.17817E-07
},
    { 156000.,   2.66270E-06,   1.08202E+03,   8.18087E-15,  -8.54029E-07
},
    { 158000.,   2.46830E-06,   1.08202E+03,   2.36086E-15,   2.28931E-07
},
    { 160000.,   2.28810E-06,   1.08202E+03,   3.67571E-15,  -6.16972E-08
},
    { 162000.,   2.12120E-06,   1.08202E+03,   2.88632E-15,   1.78573E-08
},
    { 164000.,   1.96640E-06,   1.08202E+03,   2.92903E-15,  -9.73206E-09
},
    { 166000.,   1.82300E-06,   1.08202E+03,   2.49757E-15,   2.10709E-08
},
    { 168000.,   1.69000E-06,   1.08202E+03,   2.68069E-15,  -7.45517E-08
},
```

```
        { 170000.,     1.56680E-06,   1.08202E+03,   1.47966E-15,    2.77136E-07
},
        { 172000.,     1.45250E-06,   1.08202E+03,   4.75068E-15,   -1.03399E-06
},
        { 174000.,     1.35240E-06,   1.07963E+03,   8.17622E-16,    2.73830E-07
},
        { 176000.,     1.25880E-06,   1.07723E+03,   1.72883E-15,   -7.63301E-08
},
        { 178000.,     1.17130E-06,   1.07482E+03,   1.41704E-15,    1.64901E-08
},
        { 180000.,     1.08960E-06,   1.07240E+03,   1.30299E-15,   -4.63038E-09
},
        { 182000.,     1.01320E-06,   1.06998E+03,   1.32100E-15,    2.03140E-09
},
        { 184000.,     9.41950E-07,   1.06756E+03,   1.13799E-15,   -3.49522E-09
},
        { 186000.,     8.75370E-07,   1.06513E+03,   1.13202E-15,   -3.05052E-09
},
        { 188000.,     8.13260E-07,   1.06269E+03,   1.03892E-15,    6.97283E-10
},
        { 190000.,     7.55320E-07,   1.06025E+03,   9.67290E-16,    2.61383E-10
},
        { 192000.,     7.01260E-07,   1.05781E+03,   9.11920E-16,   -1.74281E-09
},
        { 194000.,     6.50850E-07,   1.05536E+03,   8.60032E-16,   -8.29013E-09
},
        { 196000.,     6.03870E-07,   1.05290E+03,   7.92951E-16,    1.99033E-08
},
        { 198000.,     5.60110E-07,   1.05044E+03,   7.98164E-16,   -7.13232E-08
},
        { 200000.,     5.19320E-07,   1.04798E+03,   4.69394E-16,    2.65389E-07
},
        { 202000.,     4.81340E-07,   1.04550E+03,   1.53926E-15,   -1.02023E-06
},
        { 204000.,     4.47960E-07,   1.04063E+03,   2.73571E-16,    2.30547E-07
},
        { 206000.,     4.16690E-07,   1.03565E+03,   5.31456E-16,   -6.69551E-08
},
        { 208000.,     3.87320E-07,   1.03065E+03,   4.50605E-16,    7.27308E-09
},
        { 210000.,     3.59790E-07,   1.02562E+03,   4.26126E-16,   -7.13720E-09
},
        { 212000.,     3.33970E-07,   1.02057E+03,   4.09893E-16,   -8.72426E-09
},
        { 214000.,     3.09780E-07,   1.01549E+03,   3.79301E-16,   -2.96576E-09
},
        { 216000.,     2.87120E-07,   1.01039E+03,   3.67902E-16,   -9.41272E-09
},
        { 218000.,     2.65920E-07,   1.00526E+03,   3.39092E-16,   -4.38337E-09
},
        { 220000.,     2.46090E-07,   1.00011E+03,   3.30732E-16,   -3.05378E-09
},
        { 222000.,     2.27570E-07,   9.94940E+02,   3.02981E-16,   -1.34015E-08
},
        { 224000.,     2.10270E-07,   9.89730E+02,   2.87343E-16,   -3.34027E-09
},
        { 226000.,     1.94120E-07,   9.84500E+02,   2.72646E-16,   -3.23743E-09
},
        { 228000.,     1.79060E-07,   9.79250E+02,   2.57074E-16,   -1.37100E-08
},
        { 230000.,     1.65030E-07,   9.73960E+02,   2.44060E-16,   -1.92258E-09
},
        { 232000.,     1.51970E-07,   9.68650E+02,   2.21687E-16,   -8.59969E-09
},
        { 234000.,     1.39810E-07,   9.63310E+02,   2.19191E-16,   -8.67865E-09
},
        { 236000.,     1.28510E-07,   9.57940E+02,   1.91549E-16,   -1.68569E-09
},
        { 238000.,     1.18020E-07,   9.52550E+02,   2.29613E-16,   -1.45786E-08
},
        { 240000.,     1.08270E-07,   9.47120E+02,   0.00000E+00,    0.00000E+00
}
    };

    index = altitude / 2000;
    if (index > (MAX_ALT_INDEX-2))
    {
     index = MAX_ALT_INDEX-2; /* limit maximum altitude */
     altitude = MAX_ALTITUDE;
    }
    if (index < 0) index = 0;
    daltp = alt_1 - altitude;
    daltp3 = daltp*daltp*daltp;
    daltn = altitude - alt_0;
    daltn3 = daltn*daltn*daltn;

    density = (gden_0/6)*((daltp3/2000) - 2000*daltp)
                        + (gden_1/6)*((daltn3/2000) - 2000*daltn)
                        + den_0*daltp/2000 + den_1*daltn/2000;

    *v_sound = (gsps_0/6)*((daltp3/2000) - 2000*daltp)
                        + (gsps_1/6)*((daltn3/2000) - 2000*daltn)
                        + sps_0*daltp/2000 + sps_1*daltn/2000;

    *sigma = density/SEA_LEVEL_DENSITY;

    if (altitude < HLEV)    /* BUG - these curve fits are only good to about 75000 f
t */
      {
        t_amb_r = 1. - 6.875e-6*altitude;
        p_amb_r = pow( t_amb_r, 5.256 );
      }
    else
      {
        t_amb_r = 0.751895;
        p_amb_r = 0.2234*exp( -4.806e-5 * (altitude - HLEV));
      }

    *p_amb = p_amb_r * PAMB0;
    *t_amb = t_amb_r * TAMB0;

/* end of atmos_62 */
}
/*****************************************************************************/
```

```
/*************************************************************************

        TITLE:          engine.c

--------------------------------------------------------------------------

        FUNCTION:       dummy engine routine

--------------------------------------------------------------------------

        MODULE STATUS:  incomplete

--------------------------------------------------------------------------

        GENEALOGY:      Created 15 OCT 92 as dummy routine for checkout. EBJ

--------------------------------------------------------------------------

        DESIGNED BY:    designer

        CODED BY:       programmer

        MAINTAINED BY:  maintainer

--------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER INFO:

$Header: /aces/larcsim/dev/RCS/default_model_routines.c,v 1.3 1994/01/11 19:10:45 bjax S
tab $

$Log: default_model_routines.c,v $
 * Revision 1.3  1994/01/11  19:10:45  bjax
 * Removed include files.
 *
 * Revision 1.2  1993/03/14  12:16:10  bjax
 * simple correction: added ';' to end of default routines. EBJ
 *
 * Revision 1.1  93/03/10  06:43:46  bjax
 * Initial revision
 *
 * Revision 1.1  92/12/30  13:21:46  bjax
 * Initial revision
 *

--------------------------------------------------------------------------

        REFERENCES:

--------------------------------------------------------------------------

        CALLED BY:      ls_model();

--------------------------------------------------------------------------

        CALLS TO:       none

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

void inertias()         {}
void subsystems()       {}
void engine()           {}
void gear()             {}
```

```
/**************************************************************************

        TITLE:          ls_ACES.c

----------------------------------------------------------------------------

        FUNCTION:       ACES cockpit interface routines for LaRCsim

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 20 DEC 93 by E. B. Jackson for ver 1.3

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:  Bruce Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        940215  Added calibration values for stick positions; added logic to
                use calibration values; added call to ls_ACES_cal()
                routine; moved rcsid inside function body.              EBJ

        950228  Changed gear_sel_down to gear_sel_up.                   EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_ACES.c,v 1.8 1995/02/28 20:36:15 bjax Stab $
$Log: ls_ACES.c,v $
 * Revision 1.8  1995/02/28  20:36:15  bjax
 * Changed name of gear_sel_down to gear_sel_up to reflect correct
 * sense of switch. EBJ
 *
 * Revision 1.7  1994/09/01  14:55:18  bailey
 * Extended time for speedbrake deployment
 *
 *
 * Revision 1.7  1994/08/25  14:14:21  mlb
 * Throttle set to zero and speedbrake implemented for
 * reverse thrust.
 *
 * Revision 1.6  1994/05/06  21:08:10  bjax
 * Fixed glitch in channel 0, due to improper setting of CSR reg.
 *
 * Revision 1.5  1994/05/06  20:22:52  bjax
 * Changed raw[] buffer from signed short to unsigned short.
 * Added CAL_FILE macro for calibration file path and name.
 *
 * Revision 1.4  1994/04/11  20:42:07  bjax
 * Added support for rudder pedals by THRUSTMASTER!
 *
 * Revision 1.3  1994/02/16  17:36:28  bjax
 * Moved rcsid to inside function body to get rid of linker warnings.
 *
 * Revision 1.2  1994/02/16  13:08:36  bjax
 * Added calibration for stick; moved some definitions to ls_ACES.h file.
 *
 * Revision 1.1  1993/12/21  14:34:49  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <sys/resource.h>
#include <math.h>

#include "ls_types.h"
#include "ls_cockpit.h"
#include "ls_ACES.h"

#define DT 0.01666667
#define THRTAU 0.5

#define CAL_FILE "/aces/data/ACES.cal"

#define aroundup(x) ((char *)(((int) (x) + page_size - 1) & ~(page_size - 1)))
#define v_int           unsigned int *
#ifndef TRUE
#define TRUE -1
#endif
#ifndef FALSE
#define FALSE 0
#endif

struct vmic3114_control {
   unsigned short int ident_1;              /* 0x00 contains 0x000A */
   unsigned short int ident_2;              /* 0x02 contains 0x0000 */
   unsigned short int csr;                  /* 0x04 Control Status Reg */
   unsigned short int reserved_1;           /* 0x06 */
   unsigned short int ssr;                  /* 0x08 Scan Status Reg */
   unsigned short int io_ports;             /* 0x0A */
   unsigned short int icr;                  /* 0x0C Input Control Reg */
   unsigned short int ocr;                  /* 0x0E Output Control Reg */
```

```c
   unsigned short int intr_control_input;    /* 0x10 interrupt control */
   unsigned short int intr_control_output;   /* 0x12 interrupt control */
   unsigned short int reserved_2;            /* 0x14 */
   unsigned short int reserved_3;            /* 0x16 */
   unsigned short int intr_vector_input;     /* 0x18 interrupt vector */
   unsigned short int intr_vector_output;    /* 0x1A interrupt vector */
   unsigned short int reserved_4;            /* 0x1C */
   unsigned short int reserved_5;            /* 0x1E */
};

struct vmic3114_data {
   short int input_buffer[65536];         /* 64K words of input buffer */
   short int output_buffer[65536];         /* 64K words of output buffer */
};

#define LONGAD  0
#define LATAD   1
#define PEDAD   6

#define BOARD_ID              0x000A          /* Board ID code (word) */

#define FAIL_LED_OFF          0x8000          /* CSR Control bits */
#define ANALOG_OUTPUTS_ON     0x4000
#define PORT1_DIR_OUT         0x2000
#define PORT0_DIR_OUT         0x1000
#define ADVANCE_OUTPUT_ADDR 0x0800
#define OUTPUT_SINGLE_SCAN    0x0400
#define OUTPUT_SYNCHRONOUS    0x0200
#define SEL_OUTPUT_BUFF_1     0x0100
#define ADVANCE_INPUT_ADDR    0x0080
#define INPUT_SINGLE_SCAN     0x0040
#define INPUT_SYNCHRONOUS     0x0020
#define SEL_INPUT_BUFF_B      0x0010
#define SOFTWARE_RESET        0x0008
#define MASTER_MODE           0x0004
#define DIS_OUTPUT_SCAN       0x0002
#define ENABLE_BUFFERS        0x0001

#define INPUTS_ARE_DIFF       0x0080          /* SSR Status bits */
#define BINARY_CODING         0x0040
#define DAC_SCAN_COMPLETE     0X0020
#define DAC_SCAN_BUFF_A       0x0010
#define DAC_SYNC_ENABLED      0x0008
#define ADC_SCAN_COMPLETE     0x0004
#define ADC_SCAN_BUFF_A       0x0002
#define ADC_SYNC_ENABLED      0x0001

#define TEST_MODE_3           0x1800          /* ICR & OCR bit codes */
#define TEST_MODE_2           0x1000          /* not all bits apply  */
#define TEST_MODE_1           0x0800          /* to both ocr & icr   */
#define TEST_MODE_OFF         0x0000
#define GAIN_16               0x0700
#define GAIN_8                0x0300
#define GAIN_4                0x0200
#define GAIN_2                0x0100
#define GAIN_1                0x0000
#define SCAN_3                0x00F0          /* OCR ONLY!!!       */
#define SCAN_7                0x00E0          /* OCR ONLY!!!       */
#define SCAN_15               0x00D0          /* OCR ONLY!!!       */
#define SCAN_30               0x00C0          /* OCR ONLY!!!       */
#define SCAN_61               0x00B0          /* OCR ONLY!!!       */
#define SCAN_122              0x00A0          /* OCR ONLY!!!       */
#define SCAN_244              0x0090          /* OCR ONLY!!!       */
#define DIS_INPUT_SCAN        0x0080          /* <- ICR only       */
#define SCAN_488              0x0080          /* OCR ONLY!!!       */
#define SCAN_977              0x0070          /* These faster scans work for both out
and in        */
#define SCAN_1953             0x0060
#define SCAN_3906             0x0050
#define SCAN_7813             0x0040
#define SCAN_15625            0x0030
#define SCAN_31250            0x0020
#define SCAN_62500            0x0010
#define SCAN_125000           0x0000
#define BUFF_64K              0x000F
#define BUFF_32K              0x000E
#define BUFF_16K              0x000D
#define BUFF_8K               0x000C
#define BUFF_4K               0x000B
#define BUFF_2K               0x000A
#define BUFF_1K               0x0009
#define BUFF_512              0x0008
#define BUFF_256              0x0007
#define BUFF_128              0x0006
#define BUFF_64               0x0005
#define BUFF_32               0x0004
#define BUFF_16               0x0003
#define BUFF_8                0x0002
#define BUFF_4                0x0001
#define BUFF_2                0x0000

#define FLAG_BIT              0x80            /* BIM control bits */
#define FLAG_AUTO_CLEAR       0x40
#define EXTERNAL_VECTOR       0x00            /* always 0 (see manual */
#define INTERRUPT_ENABLE      0x10
#define INTR_AUTO_CLEAR       0x08
#define REQUEST_LEVEL_7       0x07
#define REQUEST_LEVEL_6       0x06
#define REQUEST_LEVEL_5       0x05
#define REQUEST_LEVEL_4       0x04
#define REQUEST_LEVEL_3       0x03
#define REQUEST_LEVEL_2       0x02
#define REQUEST_LEVEL_1       0x01
#define INTERRUPTS_OFF        0x00


/*================  VMIC INTERFACE ROUTINES ========================*/

void adc_reset(struct vmic3114_control *adc_control)
{
   adc_control->csr = 0x0008;
   sginap(50);
}

void adc_aquire(struct vmic3114_control *adc_control)
{
   adc_control->icr = 0x0012;
   adc_control->csr = 0x8005;
}

char *attmemdev(unsigned long int addr, unsigned long int size,
                char *devname)
{
   unsigned char *vme_adr;
   char *taddr;
   off_t dev_base;
   caddr_t tstart_adr;
   int fdvme32, page_size;
   v_int status;
   int rcode;
```

```
   struct rlimit *rlp;

   page_size = getpagesize();

   rlp = (struct rlimit *) malloc( sizeof( struct rlimit ));

   rcode = getrlimit( RLIMIT_DATA, rlp );
   if (rcode < 0) {
       perror("attmemdev: call to getrlimit(RLIMIT_DATA) failed ");
   }
   rcode = getrlimit( RLIMIT_STACK, rlp );
   if (rcode < 0) {
       perror("attmemdev: call to getrlimit(RLIMIT_STACK) failed ");
   }
   rcode = getrlimit( RLIMIT_VMEM, rlp );
   if (rcode < 0) {
       perror("attmemdev: call to getrlimit(RLIMIT_VMEM) failed ");
   }
   rcode = getrlimit( RLIMIT_RSS, rlp );
   if (rcode < 0) {
       perror("attmemdev: call to getrlimit(RLIMIT_RSS) failed ");
   }

   vme_adr = (unsigned char *)addr;
   dev_base = (off_t)vme_adr;

   tstart_adr = (caddr_t)sbrk(size);
   if ((int)tstart_adr == -1) {
      perror("attmem: sbrk allocation failed ");
      return ((char *)0);
   }
/*   taddr = (char *)aroundup((int)(tstart_adr)); */
    taddr = 0;  /* changed since mmap likes to put whereever if taddr = 0 */

   fdvme32 = open(devname, O_RDWR);
   if (fdvme32 < 0) {
      perror("attmem: vme i/o OPEN failed ");
      printf("device: %s\n", devname);
      return ((char *)0);
   }
   status = (v_int)mmap(taddr, size, PROT_READ | PROT_WRITE, MAP_SHARED,
                        fdvme32, dev_base);
   if ((int)status == -1) {
      perror("attmem: mmap failed ");
      return ((char *)0);
   }
   return ((char *)((unsigned long)status));
}

int adc_attach(struct vmic3114_control **adc_control,
               struct vmic3114_data **adc_data)
{
   int succ;

   succ = TRUE;

   *adc_control = (struct vmic3114_control *)attmemdev(VMIC3114_CONTROL_ADDR,
                                                       VMIC3114_CONTROL_LENGTH,
                                                       VMIC3114_CONTROL_BUS_DEVICE);
   if ((int)(*adc_control == 0)) {
      printf("adc_attach: Unable to attach to vmic3114 control store.\n");
      succ = FALSE;
   }

   if (succ) {
      *adc_data = (struct vmic3114_data *)attmemdev(VMIC3114_DATA_ADDR,
                                                    VMIC3114_DATA_LENGTH,
                                                    VMIC3114_DATA_BUS_DEVICE);
      if ((int)(*adc_data == 0)) {
         printf("adc_attach: Unable to attach to vmic3114 data block\n");
         succ = FALSE;
      }
   }

   return succ;
}

/*================  ACES COCKPIT INTERFACE ROUTINES ========================*/


ls_ACES( )
{
char rcsid[] = "$Id: ls_ACES.c,v 1.8 1995/02/28 20:36:15 bjax Stab $";

    /* calibration data */

    const static CALIBRATION_DATA cal;

  /* output limits */

    const static float tfwdstop   =  1.0;
    const static float taftstop   = -0.01;
    const static float stkfwdstop =  1.0;
    const static float stkaftstop = -1.0;
    const static float stklftstop = -1.0;
    const static float stkrgtstop =  1.0;
    const static float pedlftstop = -1.0;
    const static float pedrgtstop =  1.0;

    static  float tthrow;

    static  float trange_inv[4];

    static  float fwdscale, aftscale, lftscale, rgtscale, pedlscale, pedrscale;

    static  struct vmic3114_control *adc_control;
    static  struct vmic3114_data *adc_data;

    short   int soft_csr, soft_ssr, soft_icr;
    static  int inited = 0;

    static  float arg;
    static  float c1, c2;
    float   thrin[4];
    static  float throttle_past[4] = {0.2, 0.2, 0.2, 0.2};

    static float speedbrake;
    static float   buf[8];
    float   long_AD, lat_AD, ped_AD, long_sig, lat_sig,  ped_sig;
    static unsigned short int raw[8];
    static long wait;
    short  unsigned int di, dio;
    int i, j;

    if (!inited)
      {

        inited = -1;

        ls_ACES_cal( &cal ); /* call the calibration routine */
```

```
    /* calculate throttle filter coefficients assuming dt */

    arg =  -DT/THRTAU ;
    c2 = exp(arg);
    c1 = 1-c2;

    /* calculate scaling values */

    tthrow = tfwdstop - taftstop;

    for( i = 0; i<4; i++ )
        trange_inv[i] = 1.0/(cal.thrfwd[i] - cal.thraft[i]);

    fwdscale = stkfwdstop/(cal.stkfwd - cal.stklg0);
    aftscale = stkaftstop/(cal.stkaft - cal.stklg0);

    lftscale = stklftstop/(cal.stklft - cal.stklt0);
    rgtscale = stkrgtstop/(cal.stkrgt - cal.stklt0);

    pedlscale = pedlftstop/(cal.pedlft - cal.pedctr);
    pedrscale = pedrgtstop/(cal.pedrgt - cal.pedctr);

    if (!adc_attach(&adc_control,&adc_data)) {
      printf("Unable to connect to adc card.\n");
      exit(1);
    }

    /* reset card */

    adc_control->csr = SOFTWARE_RESET;
    do
      {
        soft_csr = adc_control->csr;
      }
    while ( soft_csr & SOFTWARE_RESET );

    /* set up for synchronous read */

    adc_control->icr = (SCAN_125000 | BUFF_8);
    adc_control->csr = (FAIL_LED_OFF | INPUT_SINGLE_SCAN
                         | MASTER_MODE | ENABLE_BUFFERS | DIS_OUTPUT_SCAN);

    soft_icr = adc_control->icr;
    adc_control->icr = soft_icr | DIS_INPUT_SCAN;  /* Put DISABLE IN hi */
    sginap(1);
    adc_control->icr = soft_icr & ~DIS_INPUT_SCAN;  /* Put DISABLE IN lo */

    /* set up to read all 16 inputs */

    adc_control->csr &= ((~PORT1_DIR_OUT) & (~PORT0_DIR_OUT));

  }

/* Normal run loop cycles through here */

wait = 0;
do
  {
    soft_ssr = adc_control->ssr;
    wait++;
  }
while ((soft_ssr & ADC_SCAN_BUFF_A) ); /* wait for AD SCAN A */

soft_icr = adc_control->icr;          /* Start of single-scan toggle */
adc_control->icr = soft_icr | DIS_INPUT_SCAN;  /* Put DISABLE IN hi */

for( i=0; i<8; i++)
{
    raw[i] = adc_data->input_buffer[i];
    buf[i] = ((float)(raw[i])/204.8) - 10.0;
}

di = ~(adc_control->io_ports);

/* debugging code to display DI's
 *
 *
    j=0;
    dio = di;
    for (i=0;i<16;i++)
      {
        if( dio & 0x8000 ) printf("1"); else printf("0");
        dio = dio << 1;
        j++;
        if (j >= 4)
          {
            j = 0;
            printf(" ");
          }
      }
    printf("\n");
 */

long_AD = buf[LONGAD];
lat_AD  = buf[LATAD];
ped_AD  = buf[PEDAD];

long_sig = long_AD - cal.stklg0;
lat_sig  = lat_AD  - cal.stklt0;
ped_sig  = ped_AD  - cal.pedctr;

SB_extend = di & SPEEDBRAKE_EXTEND;
SB_retract = di & SPEEDBRAKE_RETRACT;

speedbrake=speedbrake-(.00002*SB_extend)+(.00002*SB_retract);
if(speedbrake > 0.)   speedbrake = 0.;
if(speedbrake < -0.2) speedbrake = -0.2;

if (long_sig > 0.)
    Long_control = long_sig*fwdscale;
else
    Long_control = long_sig*aftscale;

if (lat_sig > 0.)
    Lat_control = lat_sig*rgtscale;
else
    Lat_control = lat_sig*lftscale;

if (ped_sig > 0.)
    Rudder_pedal = ped_sig*pedrscale;
else
    Rudder_pedal = ped_sig*pedlscale;

for( i=0; i<4; i++)
{
    thrin[i] = (float) tthrow*((float) buf[i+2]
        - cal.thraft[i])*trange_inv[i] + taftstop +speedbrake;

    if (thrin[i] > 1.) thrin[i] = 1.;
```

```
        if (thrin[i] < -0.2) thrin[i] = -0.2;

        Throttle[i] = thrin[i]*c1 + throttle_past[i]*c2;
        throttle_past[i] = Throttle[i];
    }

    Fwd_trim = di & FWD_TRIM;
    Aft_trim = di & AFT_TRIM;
    Right_trim  = di & RIGHT_TRIM;
    Left_trim  = di & LEFT_TRIM;

    Left_button = di & LEFT_PUSH_BUTTON;
    Right_button = di & RIGHT_PUSH_BUTTON;
    First_trigger = di & FIRST_TRIG_BUTTON;
    Second_trigger = di & SECOND_TRIG_BUTTON;

    Gear_sel_up = di & LANDING_GEAR_UP;

    adc_control->icr = soft_icr & ~DIS_INPUT_SCAN;  /* Put DISABLE IN lo to start scan *
/
}


ls_ACES_cal( CALIBRATION_DATA *cal )    /* performs cockpit calibration routine */

{
    FILE *fp;
    int num;

    fp = fopen(CAL_FILE, "r");
    if (fp != 0)
    /* if ACES.cal exists, use it */
        {
            num = fscanf( fp, "%f %f %f %f\n",
                &cal->thraft[0],  &cal->thraft[1], &cal->thraft[2], &cal->thraft[3] );
            if (num != 4) fprintf(stderr, "ACES.cal improper format, line 1\n");
            num = fscanf( fp, "%f %f %f %f\n",
                &cal->thrfwd[0],  &cal->thrfwd[1], &cal->thrfwd[2], &cal->thrfwd[3] );
            if (num != 4) fprintf(stderr, "ACES.cal improper format, line 2\n");
            num = fscanf( fp, "%f %f %f\n",
                &cal->stkaft, &cal->stklg0, &cal->stkfwd );
            if (num != 3) fprintf(stderr, "ACES.cal improper format, line 3\n");
            num = fscanf( fp, "%f %f %f\n",
                &cal->stklft, &cal->stklt0, &cal->stkrgt );
            if (num != 3) fprintf(stderr, "ACES.cal improper format, line 4\n");
            num = fscanf( fp, "%f %f %f\n",
                &cal->pedlft, &cal->pedctr, &cal->pedrgt );
            if (num != 3) fprintf(stderr, "ACES.cal improper format, line 5\n");
            fclose( fp );
        }
    else
    /* use default calibration numbers */   /* 931215  EBJ */
        {
            fprintf(stderr,  CAL_FILE);
            fprintf(stderr, " not found; using default values\n");
            cal->thraft[0] =  -5.752;
            cal->thraft[1] =  -6.709;
            cal->thraft[2] =  -6.548;
            cal->thraft[3] =  -7.314;
            cal->thrfwd[0] =  -3.325;
            cal->thrfwd[1] =  -4.307;
            cal->thrfwd[2] =  -4.097;
            cal->thrfwd[3] =  -4.858;
            cal->stkaft    = -10.000;
            cal->stklg0    =  -0.405;
            cal->stkfwd    =   9.116;
            cal->stklft    =  -9.996;
            cal->stklt0    =  -0.366;
            cal->stkrgt    =   9.297;
            cal->pedlft    =  -5.900;
            cal->pedctr    =  -0.100;
            cal->pedrgt    =   5.600;
        }

}
```

```
/***************************************************************************

  TITLE:        ls_Accel

 ----------------------------------------------------------------------------

  FUNCTION:     Sums forces and moments and calculates accelerations

 ----------------------------------------------------------------------------

  MODULE STATUS:        developmental

 ----------------------------------------------------------------------------

  GENEALOGY:    Written 920731 by Bruce Jackson.  Based upon equations
  given in reference [1] and a Matrix-X/System Build block
  diagram model of equations of motion coded by David Raney
  at NASA-Langley in June of 1992.

 ----------------------------------------------------------------------------

  DESIGNED BY:  Bruce Jackson

  CODED BY:             Bruce Jackson

  MAINTAINED BY:

 ----------------------------------------------------------------------------

  MODIFICATION HISTORY:

  DATE          PURPOSE

  931007    Moved calculations of auxiliary accelerations here from ls_aux.c
BY
            and corrected minus sign in front of A_Y_Pilot
            contribution from Q_body*P_body*D_X_pilot term.
  940111    Changed DATA to SCALAR; updated header files

$Header: /aces/larcsim/dev/RCS/ls_accel.c,v 1.5 1994/01/11 18:42:16 bjax Stab $
$Log: ls_accel.c,v $
 * Revision 1.5  1994/01/11  18:42:16  bjax
 * Oops! Changed data types from DATA to SCALAR for precision control.
 *
 * Revision 1.4  1994/01/11  18:36:58  bjax
 * Removed ls_eom.h include directive; replaced with ls_types, ls_constants,
 * and ls_generic.h includes.
 *
 * Revision 1.3  1993/10/07  18:45:24  bjax
 * Added local defn of d[xyz]_pilot_from_cg to support previous mod. EBJ
 *
 * Revision 1.2  1993/10/07  18:41:31  bjax
 * Moved calculations of auxiliary accelerations here from ls_aux, and
 * corrected sign on Q_body*P_body*d_x_pilot term of A_Y_pilot calc.  EBJ
 *
 * Revision 1.1  1992/12/30  13:17:02  bjax
 * Initial revision
 *

 ----------------------------------------------------------------------------

  REFERENCES:

  [ 1] McFarland, Richard E.: "A Standard Kinematic Model
  for Flight Simulation at NASA-Ames", NASA CR-2497,
  January 1975

 ----------------------------------------------------------------------------

  CALLED BY:

 ----------------------------------------------------------------------------

  CALLS TO:

 ----------------------------------------------------------------------------

  INPUTS:  Aero, engine, gear forces & moments

 ----------------------------------------------------------------------------

  OUTPUTS:      State derivatives

 --------------------------------------------------------------------------*/
#include "ls_types.h"
#include "ls_generic.h"
#include "ls_constants.h"
#include <math.h>

void ls_accel(  )
{

  SCALAR        inv_Mass, inv_Radius;
  SCALAR        ixz2, c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10;
  SCALAR        dx_pilot_from_cg, dy_pilot_from_cg, dz_pilot_from_cg;


  /* Sum forces and moments at reference point */


  F_X = F_X_aero + F_X_engine + F_X_gear;
  F_Y = F_Y_aero + F_Y_engine + F_Y_gear;
  F_Z = F_Z_aero + F_Z_engine + F_Z_gear;

  M_l_rp = M_l_aero + M_l_engine + M_l_gear;
  M_m_rp = M_m_aero + M_m_engine + M_m_gear;
  M_n_rp = M_n_aero + M_n_engine + M_n_gear;

  /* Transfer moments to center of gravity */

  M_l_cg = M_l_rp + F_Y*Dz_cg - F_Z*Dy_cg;
  M_m_cg = M_m_rp + F_Z*Dx_cg - F_X*Dz_cg;
  M_n_cg = M_n_rp + F_X*Dy_cg - F_Y*Dx_cg;

  /* Transform from body to local frame */

  F_north = T_local_to_body_11*F_X + T_local_to_body_21*F_Y
    + T_local_to_body_31*F_Z;
  F_east  = T_local_to_body_12*F_X + T_local_to_body_22*F_Y
    + T_local_to_body_32*F_Z;
  F_down  = T_local_to_body_13*F_X + T_local_to_body_23*F_Y
    + T_local_to_body_33*F_Z;

  /* Calculate linear accelerations */

  inv_Mass = 1/Mass;
  inv_Radius = 1/Radius_to_vehicle;
  V_dot_north = inv_Mass*F_north +
    inv_Radius*(V_north*V_down - V_east*V_east*tan(Lat_geocentric));
  V_dot_east  = inv_Mass*F_east  +
```

```
    inv_Radius*(V_east*V_down + V_north*V_east*tan(Lat_geocentric));
  V_dot_down  = inv_Mass*(F_down) + Gravity -
    inv_Radius*(V_north*V_north + V_east*V_east);

  /* Invert the symmetric inertia matrix */

  ixz2 = I_xz*I_xz;
  c0  = 1/(I_xx*I_zz - ixz2);
  c1  = c0*((I_yy-I_zz)*I_zz - ixz2);
  c2  = c0*I_xz*(I_xx - I_yy + I_zz);
  c3  = c0*I_zz;
  c4  = c0*I_xz;
  c7  = 1/I_yy;
  c5  = c7*(I_zz - I_xx);
  c6  = c7*I_xz;
  c8  = c0*((I_xx - I_yy)*I_xx + ixz2);
  c9  = c0*I_xz*(I_yy - I_zz - I_xx);
  c10 = c0*I_xx;

  /* Calculate the rotational body axis accelerations */

  P_dot_body = (c1*R_body + c2*P_body)*Q_body + c3*M_l_cg +  c4*M_n_cg;
  Q_dot_body = c5*R_body*P_body + c6*(R_body*R_body - P_body*P_body) + c7*M_m_cg;
  R_dot_body = (c8*P_body + c9*R_body)*Q_body + c4*M_l_cg + c10*M_n_cg;

  /* Calculate body axis accelerations (move to ls_accel?)      */

    inv_Mass = 1/Mass;

    A_X_cg = F_X * inv_Mass;
    A_Y_cg = F_Y * inv_Mass;
    A_Z_cg = F_Z * inv_Mass;

    dx_pilot_from_cg = Dx_pilot - Dx_cg;
    dy_pilot_from_cg = Dy_pilot - Dy_cg;
    dz_pilot_from_cg = Dz_pilot - Dz_cg;

    A_X_pilot = A_X_cg  + (-R_body*R_body - Q_body*Q_body)*dx_pilot_from_cg
                                          + ( P_body*Q_body - R_dot_body   )*dy_pilot_
from_cg
                                          + ( P_body*R_body + Q_dot_body   )*dz_pilot_
from_cg;
    A_Y_pilot = A_Y_cg  + ( P_body*Q_body + R_dot_body   )*dx_pilot_from_cg
                                          + (-P_body*P_body - R_body*R_body)*dy_pilot_
from_cg
                                          + ( Q_body*R_body - P_dot_body   )*dz_pilot_
from_cg;
    A_Z_pilot = A_Z_cg  + ( P_body*R_body - Q_dot_body   )*dx_pilot_from_cg
                                          + ( Q_body*R_body + P_dot_body   )*dy_pilot_
from_cg
                                          + (-Q_body*Q_body - P_body*P_body)*dz_pilot_
from_cg;

    N_X_cg = INVG*A_X_cg;
    N_Y_cg = INVG*A_Y_cg;
    N_Z_cg = INVG*A_Z_cg;

    N_X_pilot = INVG*A_X_pilot;
    N_Y_pilot = INVG*A_Y_pilot;
    N_Z_pilot = INVG*A_Z_pilot;


    U_dot_body = T_local_to_body_11*V_dot_north + T_local_to_body_12*V_dot_east
                                  + T_local_to_body_13*V_dot_down - Q_total*W_body + R
_total*V_body;
    V_dot_body = T_local_to_body_21*V_dot_north + T_local_to_body_22*V_dot_east
                                  + T_local_to_body_23*V_dot_down - R_total*U_body
 + P_total*W_body;
    W_dot_body = T_local_to_body_31*V_dot_north + T_local_to_body_32*V_dot_east
                                  + T_local_to_body_33*V_dot_down - P_total*V_body
 + Q_total*U_body;
    /* End of ls_accel */

}
/**************************************************************************/
```

```
/*************************************************************************

    TITLE:          ls_aux

--------------------------------------------------------------------------

    FUNCTION:   Atmospheric and auxilary relationships for LaRCSim EOM

--------------------------------------------------------------------------

    MODULE STATUS:      developmental

--------------------------------------------------------------------------

    GENEALOGY:  Created 9208026 as part of C-castle simulation project.

--------------------------------------------------------------------------

    DESIGNED BY:        B. Jackson

    CODED BY:           B. Jackson

    MAINTAINED BY:      B. Jackson

--------------------------------------------------------------------------

    MODIFICATION HISTORY:

    DATE    PURPOSE

    931006  Moved calculations of auxiliary accelerations from here
            to ls_accel.c and corrected minus sign in front of A_Y_Pilot
            contribution from Q_body*P_body*D_X_pilot term.        EBJ
    931014  Changed calculation of Alpha from atan to atan2 so sign is correct.
                                                                   EBJ
    931220  Added calculations for static and total temperatures & pressures,
            as well as dynamic and impact pressures and calibrated airspeed.
                                                                   EBJ
    940111  Changed #included header files from old "ls_eom.h" to newer
            "ls_types.h", "ls_constants.h" and "ls_generic.h".     EBJ

    950207  Changed use of "abs" to "fabs" in calculation of signU. EBJ

    950228  Fixed bug in calculation of beta_dot.                  EBJ

    CURRENT RCS HEADER INFO:

$Header: /aces/larcsim/dev/RCS/ls_aux.c,v 1.12 1995/02/28 17:57:16 bjax Stab $
$Log: ls_aux.c,v $
 * Revision 1.12  1995/02/28  17:57:16  bjax
 * Corrected calculation of beta_dot. EBJ
 *
 * Revision 1.11  1995/02/07  21:09:47  bjax
 * Corrected calculation of "signU"; was using divide by
 * abs(), which returns an integer; now using fabs(), which
 * returns a double.  EBJ
 *
 * Revision 1.10  1994/05/10  20:09:42  bjax
 * Fixed a major problem with dx_pilot_from_cg, etc. not being calculated locally.
 *
 * Revision 1.9  1994/01/11  18:44:33  bjax
 * Changed header files to use ls_types, ls_constants, and ls_generic.
 *
 * Revision 1.8  1993/12/21  14:36:33  bjax
 * Added calcs of pressures, temps and calibrated airspeeds.
 *
 * Revision 1.7  1993/10/14  11:25:38  bjax
 * Changed calculation of Alpha to use 'atan2' instead of 'atan' so alphas
 * larger than +/- 90 degrees are calculated correctly.               EBJ
 *
 * Revision 1.6  1993/10/07  18:45:56  bjax
 * A little cleanup; no significant changes. EBJ
 *
 * Revision 1.5  1993/10/07  18:42:22  bjax
 * Moved calculations of auxiliary accelerations here from ls_aux, and
 * corrected sign on Q_body*P_body*d_x_pilot term of A_Y_pilot calc.  EBJ
 *
 * Revision 1.4  1993/07/16  18:28:58  bjax
 * Changed call from atmos_62 to ls_atmos. EBJ
 *
 * Revision 1.3  1993/06/02  15:02:42  bjax
 * Changed call to geodesy calcs from ls_geodesy to ls_geoc_to_geod.
 *
 * Revision 1.1  92/12/30  13:17:39  bjax
 * Initial revision
 *

--------------------------------------------------------------------------

    REFERENCES: [ 1] Shapiro, Ascher H.: "The Dynamics and Thermodynamics
                     of Compressible Fluid Flow", Volume I, The Ronald
                     Press, 1953.

--------------------------------------------------------------------------

                CALLED BY:

--------------------------------------------------------------------------

                CALLS TO:

--------------------------------------------------------------------------

                INPUTS:

--------------------------------------------------------------------------

                OUTPUTS:

--------------------------------------------------------------------------*/
#include "ls_types.h"
#include "ls_constants.h"
#include "ls_generic.h"
#include <math.h>

void    ls_aux()
{

        SCALAR  dx_pilot_from_cg, dy_pilot_from_cg, dz_pilot_from_cg;
        SCALAR  inv_Mass;
        SCALAR  v_XZ_plane_2, signU, v_tangential;
        SCALAR  inv_radius_ratio;
        SCALAR  cos_rwy_hdg, sin_rwy_hdg;
        SCALAR  mach2, temp_ratio, pres_ratio;

    /* update geodetic position */

        ls_geoc_to_geod( Lat_geocentric, Radius_to_vehicle,
                                &Latitude, &Altitude, &Sea_level_radius );
```

```
    Longitude = Lon_geocentric - Earth_position_angle;

/* Calculate body axis velocities */

    /* Form relative velocity vector */

    V_north_rel_ground = V_north;
    V_east_rel_ground  = V_east
      - OMEGA_EARTH*Sea_level_radius*cos( Lat_geocentric );
    V_down_rel_ground  = V_down;

    V_north_rel_airmass = V_north_rel_ground - V_north_airmass;
    V_east_rel_airmass  = V_east_rel_ground  - V_east_airmass;
    V_down_rel_airmass  = V_down_rel_ground  - V_down_airmass;

    U_body = T_local_to_body_11*V_north_rel_airmass
      + T_local_to_body_12*V_east_rel_airmass
        + T_local_to_body_13*V_down_rel_airmass + U_gust;
    V_body = T_local_to_body_21*V_north_rel_airmass
      + T_local_to_body_22*V_east_rel_airmass
        + T_local_to_body_23*V_down_rel_airmass + V_gust;
    W_body = T_local_to_body_31*V_north_rel_airmass
      + T_local_to_body_32*V_east_rel_airmass
        + T_local_to_body_33*V_down_rel_airmass + W_gust;

    V_rel_wind = sqrt(U_body*U_body + V_body*V_body + W_body*W_body);


/* Calculate alpha and beta rates   */

    v_XZ_plane_2 = (U_body*U_body + W_body*W_body);

    if (U_body == 0)
            signU = 1;
    else
            signU = U_body/fabs(U_body);

    if( (v_XZ_plane_2 == 0) || (V_rel_wind == 0) )
    {
            Alpha_dot = 0;
            Beta_dot = 0;
    }
    else
    {
            Alpha_dot = (U_body*W_dot_body - W_body*U_dot_body)/
              v_XZ_plane_2;
            Beta_dot = (signU*v_XZ_plane_2*V_dot_body
              - V_body*(U_body*U_dot_body + W_body*W_dot_body))
                /(V_rel_wind*V_rel_wind*sqrt(v_XZ_plane_2));
    }

/* Calculate flight path and other flight condition values */

    if (U_body == 0)
            Alpha = 0;
    else
            Alpha = atan2( W_body, U_body );

    Cos_alpha = cos(Alpha);
    Sin_alpha = sin(Alpha);

    if (V_rel_wind == 0)
            Beta = 0;
    else
            Beta = asin( V_body/ V_rel_wind );

    Cos_beta = cos(Beta);
    Sin_beta = sin(Beta);

    V_true_kts = V_rel_wind * V_TO_KNOTS;

    V_ground_speed = sqrt(V_north_rel_ground*V_north_rel_ground
                          + V_east_rel_ground*V_east_rel_ground );

    V_rel_ground = sqrt(V_ground_speed*V_ground_speed
                        + V_down_rel_ground*V_down_rel_ground );

    v_tangential = sqrt(V_north*V_north + V_east*V_east);

    V_inertial = sqrt(v_tangential*v_tangential + V_down*V_down);

    if( (V_ground_speed == 0) && (V_down == 0) )
      Gamma_vert_rad = 0;
    else
      Gamma_vert_rad = atan2( -V_down, V_ground_speed );

    if( (V_north_rel_ground == 0) && (V_east_rel_ground == 0) )
      Gamma_horiz_rad = 0;
    else
      Gamma_horiz_rad = atan2( V_east_rel_ground, V_north_rel_ground );

    if (Gamma_horiz_rad < 0)
      Gamma_horiz_rad = Gamma_horiz_rad + 2*PI;

/* Calculate local gravity  */

    ls_gravity( Radius_to_vehicle, Lat_geocentric, &Gravity );

/* call function for (smoothed) density ratio, sonic velocity, and
       ambient pressure */

    ls_atmos(Altitude, &Sigma, &V_sound,
             &Static_temperature, &Static_pressure);

    Density = Sigma*SEA_LEVEL_DENSITY;

    Mach_number = V_rel_wind / V_sound;

    V_equiv = V_rel_wind*sqrt(Sigma);

    V_equiv_kts = V_equiv*V_TO_KNOTS;

/* calculate temperature and pressure ratios (from [1]) */

    mach2 = Mach_number*Mach_number;
    temp_ratio = 1.0 + 0.2*mach2;
    pres_ratio = pow( temp_ratio, 3.5 );

    Total_temperature = temp_ratio*Static_temperature;
    Total_pressure    = pres_ratio*Static_pressure;

/* calculate impact and dynamic pressures */

    Impact_pressure = Total_pressure - Static_pressure;

    Dynamic_pressure = 0.5*Density*V_rel_wind*V_rel_wind;

/* calculate calibrated airspeed indication */

    V_calibrated = sqrt( 2.0*Dynamic_pressure / SEA_LEVEL_DENSITY );
```

```
        V_calibrated_kts = V_calibrated*V_TO_KNOTS;

        Centrifugal_relief = 1 - v_tangential/(Radius_to_vehicle*Gravity);

/* Determine location in runway coordinates */

        Radius_to_rwy = Sea_level_radius + Runway_altitude;
        cos_rwy_hdg = cos(Runway_heading*DEG_TO_RAD);
        sin_rwy_hdg = sin(Runway_heading*DEG_TO_RAD);

        D_cg_north_of_rwy = Radius_to_rwy*(Latitude - Runway_latitude);
        D_cg_east_of_rwy = Radius_to_rwy*cos(Runway_latitude)
                *(Longitude - Runway_longitude);
        D_cg_above_rwy  = Radius_to_vehicle - Radius_to_rwy;

        X_cg_rwy = D_cg_north_of_rwy*cos_rwy_hdg
          + D_cg_east_of_rwy*sin_rwy_hdg;
        Y_cg_rwy =-D_cg_north_of_rwy*sin_rwy_hdg
          + D_cg_east_of_rwy*cos_rwy_hdg;
        H_cg_rwy = D_cg_above_rwy;

        dx_pilot_from_cg = Dx_pilot - Dx_cg;
        dy_pilot_from_cg = Dy_pilot - Dy_cg;
        dz_pilot_from_cg = Dz_pilot - Dz_cg;

        D_pilot_north_of_rwy = D_cg_north_of_rwy
          + T_local_to_body_11*dx_pilot_from_cg
            + T_local_to_body_21*dy_pilot_from_cg
              + T_local_to_body_31*dz_pilot_from_cg;
        D_pilot_east_of_rwy  = D_cg_east_of_rwy
          + T_local_to_body_12*dx_pilot_from_cg
            + T_local_to_body_22*dy_pilot_from_cg
              + T_local_to_body_32*dz_pilot_from_cg;
        D_pilot_above_rwy    = D_cg_above_rwy
          - T_local_to_body_13*dx_pilot_from_cg
            - T_local_to_body_23*dy_pilot_from_cg
              - T_local_to_body_33*dz_pilot_from_cg;

        X_pilot_rwy =  D_pilot_north_of_rwy*cos_rwy_hdg
          + D_pilot_east_of_rwy*sin_rwy_hdg;
        Y_pilot_rwy = -D_pilot_north_of_rwy*sin_rwy_hdg
          + D_pilot_east_of_rwy*cos_rwy_hdg;
        H_pilot_rwy =  D_pilot_above_rwy;

/* Calculate Euler rates */

        Sin_phi = sin(Phi);
        Cos_phi = cos(Phi);
        Sin_theta = sin(Theta);
        Cos_theta = cos(Theta);
        Sin_psi = sin(Psi);
        Cos_psi = cos(Psi);

        if( Cos_theta == 0 )
          Psi_dot = 0;
        else
          Psi_dot = (Q_total*Sin_phi + R_total*Cos_phi)/Cos_theta;

        Theta_dot = Q_total*Cos_phi - R_total*Sin_phi;
        Phi_dot = P_total + Psi_dot*Sin_theta;

/* end of ls_aux */

}
/*************************************************************************/
```

```
/*************************************************************************

        TITLE:          ls_err.c

-------------------------------------------------------------------------

        FUNCTION:       Error reporting routines

-------------------------------------------------------------------------

        MODULE STATUS:  Developmental

-------------------------------------------------------------------------

        GENEALOGY:      Written 9112 to support Mex files; installed as
                        part of LaRCSim software

-------------------------------------------------------------------------

        DESIGNED BY:    B. Jackson

        CODED BY:       B. Jackson

        MAINTAINED BY:  B. Jackson

-------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        940106  Redirected error output to "stderr"             EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_err.c,v 1.2 1994/01/11 18:25:24 bjax Stab $
$Log: ls_err.c,v $
 * Revision 1.2  1994/01/11  18:25:24  bjax
 * Redirected output to stderr from stdout.
 *
 * Revision 1.1  1993/03/19  07:22:27  bjax
 * Initial revision
 *

-------------------------------------------------------------------------

        REFERENCES:

-------------------------------------------------------------------------

        CALLED BY:

-------------------------------------------------------------------------

        CALLS TO:

-------------------------------------------------------------------------

        INPUTS:

-------------------------------------------------------------------------

        OUTPUTS:

-------------------------------------------------------------------------*/
```

```
#include "ls_err.h"
#include <string.h>
#include <stdio.h>

#define NO_MSG "No message."


ERROR  error = {  info,
          E_NO_ERROR,
          0,
          0,
          0., 0., 0.,
          0, 0, 0        };

char  *report_error(  ERROR  *err_block )

{
  char  error_msg[ ERROR_STRING_LENGTH ];
  char  *err_msg_p;
  float  *flt_p;
  int    *int_p;

  err_msg_p = &error_msg[0];
  while (*err_msg_p++ = *err_block->strg1++)
  {
    switch (*err_block->strg1)
    {
      case '&':  /* place to insert parameter */
      {
        (err_block->strg1)++;    /* skip over '&' sign */
        switch (*err_block->strg1)  /* and evaluate next char */
        {
          case 'f':  /* float parameters */
          {
            (err_block->strg1)++;  /* skip past 'f' char */
            switch (*err_block->strg1)  /* and eval next  */
            {
              case '1':
              {  /* convert param fp1 to string */
                flt_p = &err_block->fp1;
                break;
              }
              case '2':
              {  /* convert param fp2 to string */
                flt_p = &err_block->fp2;
                break;
              }
              case '3':
              {  /* convert param fp3 to string */
                flt_p = &err_block->fp3;
                break;
              }
              default:
                /* print error message - not '&f[123]' */;
            }
            err_msg_p = err_msg_p + (int)sprintf( err_msg_p, "%e", *flt_p );
            err_block->strg1++;
            break;
          }
          case 'i':  /* Integer parameter */
          {
            (err_block->strg1)++;  /* skip past 'i' char */
            switch (*err_block->strg1)  /* and eval next  */
            {
```

```
          case '1':
          {  /* convert param ip1 to string */;
            int_p = &err_block->ip1;
            break;
          }
          case '2':
          {  /* convert param ip2 to string */;
            int_p = &err_block->ip2;
            break;
          }
          case '3':
          {  /* convert param ip3 to string */;
            int_p = &err_block->ip3;
            break;
          }
          default:
            /* print error message - not '&i[123]' */;
        }
        err_msg_p = err_msg_p + (int)sprintf( err_msg_p, "%d", *int_p );
        err_block->strg1++;
        break;
      }
      case 's':  /* string parameter */
      {
        (err_block->strg1)++;  /* skip past 's' char */
        switch (*err_block->strg1)  /* and eval next  */
        {
          case '1':  /* error - can only be a 2 */
            /* convert param fp1 to string */;
          case '2':
          {  /* convert param fp2 to string */;
            err_msg_p = err_msg_p + (int)sprintf( err_msg_p, err_block->strg2 );
            err_block->strg1++;
            break;
          }
          case '3':  /* error - can only be a 2 */
            /* convert param fp3 to string */;
          default:
            /* print error message - not '&s[123]' */;
        }
        break;
      }
      default:  /* error - ampersand alone */
      {
        /* print error message */
      }
    }
  }
}
}
fprintf(stderr, "%s",&error_msg[0]);
return &error_msg[0];
}
```

```
/***************************************************************************

        TITLE:          ls_funcgen.c

----------------------------------------------------------------------------

        FUNCTION:       Function generation routines for LaRCsim models

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:

        Function table interpolation routines written 911220 E. B. Jackson
        to support MATLAB/SIMULAB non-linear models.

        THEORY:

        Breakpoint data sets and function tables are stored separately in
        BREAKPOINTS and DATA structures. They are associated together in
        an individual FUNC_DATA structure; the FUNC_DATA structure is an
        abstraction of a multi-dimensional curve or surface.

        The NONLINEAR_FUNCTION structure associates this function data with
        the interpolation information (index and weights as well as the last
        value returned on the previous lookup call). This structure is an
        abstraction of the process of interpolating a FUNC_DATA curve; it
        includes a pointer to the function data as well as state
        information about where the function was most recently found,
        which speeds up subsequent searches, since a crawl through the
        breakpoint vector is used instead of a binary search.

        The tables are effectively unlimited in size and number of dimensions;
        the maximum length in any dimension is set by MAX_LENGTH, and the
        number of dimensions is set by MAX_DIMENSION; both are declared in
        ls_funcgen.h header file.

        Another data structure, ARG_LIST, is used to pass
        interpolation information to the lookup function. It contains
        the current index value and interpolation ratio for each
        dimension of the nonlinear function.

        USE:

        ===== Initialization process =====

        * declare parameters

        %define N_ALPHA1        10
        %define N_XMACH1         4
        %define N_DWF            5

        * declare breakpoints

        static BREAKPOINTS ALPHA1_PTS   = { "ALPHA1", N_ALPHA1,
                { 0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0 }};
        static BREAKPOINTS XMACH1_PTS   = { "XMACH1", N_XMACH1,
                { 0.3, 0.6, 0.8, 0.9 }};
        static BREAKPOINTS DWF_PTS      = { "DWF",     N_DWF,
                {  -30., -15., 0., 15., 30.}};

        * declare data set (first variable changes most rapidly)

static  SCALAR  CLWF_DATA[ N_ALPHA1*N_XMACH1*N_DWF ] = {
*       CLWF    POINTS  *
* WF =  -30.0   *
* XMACH = 0.3   *
-0.52853E-01 ,-0.55962E-01 ,-0.65289E-01 ,-0.68398E-01 ,-0.54407E-01 ,
-0.47770E-01 ,-0.43343E-01 ,-0.25638E-01 ,-0.13147E-01 ,-0.86206E-02 ,
* XMACH = 0.6   *
-0.34528E-01 ,-0.34432E-01 ,-0.35534E-01 ,-0.38899E-01 ,-0.40023E-01 ,
-0.42346E-01 ,-0.45678E-01 ,-0.57322E-01 ,-0.60051E-01 ,-0.47768E-01 ,
* XMACH = 0.8   *
-0.52853E-01 ,-0.55962E-01 ,-0.65289E-01 ,-0.68398E-01 ,-0.54407E-01 ,
-0.47770E-01 ,-0.43343E-01 ,-0.25638E-01 ,-0.13147E-01 ,-0.86206E-02 ,
* XMACH = 0.9   *
-0.34528E-01 ,-0.34432E-01 ,-0.35534E-01 ,-0.38899E-01 ,-0.40023E-01 ,
-0.42346E-01 ,-0.45678E-01 ,-0.57322E-01 ,-0.60051E-01 ,-0.47768E-01 ,

* WF =  -15.0   *
* XMACH = 0.3   *
-0.46403E-01 ,-0.49133E-01 ,-0.57322E-01 ,-0.60051E-01 ,-0.47768E-01 ,
-0.39377E-01 ,-0.33779E-01 ,-0.11394E-01 ,-0.46116E-02 ,-0.33766E-02 ,
        .
        .
        .
* WF =   30.0   *
        .
        .
        .
* XMACH = 0.9   *
0.59318E-01 , 0.87301E-01 , 0.77601E-01 , 0.48501E-01 , 0.77601E-01 ,
0.62989E-01 , 0.53241E-01 , 0.14262E-01 , 0.18469E-01 , 0.11304E-01 };


* associate the break points with function data

static FUNC_DATA CLWF_PTS = {
        "CLWF POINTS",                          * name of point set
        3,                                      * number of breakpoints
        {N_ALPHA_1, N_XMACH1, N_DWF},           * size of each breakpoint se
t
};


* a typical function will then look like this  (creation of these
* should be automated in the near future )


float   clwfl_fn( float alpha, float mach, float dwfl )

{
  static NONLINEAR_FUNCTION     CLWFL_NLF =
    { "WFL Lift", NULL, { NULL, NULL, NULL },
    { 0.0, 0.0, 0.0 }, { 4.0, 0.3, -30. } };

  ARG_LIST   arg_list;

  static int init=0;

  if (!init)
    {
        init = -1;
        CLWFL_NLF.ptr_to_data = &CLWF_PTS;
        CLWFL_NLF.bkPtList[0] = &ALPHA1_PTS;
        CLWFL_NLF.bkPtList[1] = &XMACH1_PTS;
        CLWFL_NLF.bkPtList[2] = &DWF_PTS;
```

```
        }

     * Normalize breakpoints *

            arg_list.index_and_weight[0] = normalize_bkpt( &CLWFL_NLF, 0, alpha );
            arg_list.index_and_weight[1] = normalize_bkpt( &CLWFL_NLF, 1, mach );
            arg_list.index_and_weight[2] = normalize_bkpt( &CLWFL_NLF, 2, dwfl );

     * Perform lookup and return     *

     return funcgen( &CLWFL0_NLF, &arg_list, 2 );

   } * End of CLwfl *

     ===== Operation =====

     clwfl = clwfl_fn( Alpha, Mach, Dwfl );

--------------------------------------------------------------------------

     DESIGNED BY:    Bruce Jackson

     CODED BY:       Bruce Jackson

     MAINTAINED BY:

--------------------------------------------------------------------------

     MODIFICATION HISTORY:

     DATE    PURPOSE                                              BY

     940216  Moved rcsid variable inside the function to get rid of
             archive and linker warnings.                         EBJ

     CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_funcgen.c,v 1.6 1994/05/20 21:49:03 bjax Stab $
$Log: ls_funcgen.c,v $
 * Revision 1.6  1994/05/20  21:49:03  bjax
 * Added end-of-line character to emsg1 in routine getdata.
 *
 * Revision 1.5  1994/02/16  17:34:33  bjax
 * Moved rcsid to inside function to get rid of linker warnings.
 *
 * Revision 1.4  1994/01/11  18:25:52  bjax
 * Added large amounts of comments to header record to show how to
 * use funcgen stuff.
 *

--------------------------------------------------------------------------

     REFERENCES:

--------------------------------------------------------------------------

     CALLED BY:

--------------------------------------------------------------------------

     CALLS TO:

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#include "ls_funcgen.h"
#include "ls_err.h"

extern ERROR error;

float normalize_bkpt( NONLINEAR_FUNCTION *nlfunct, int dim, DATA value )

{
#define NLFbpl  nlfunct->bkPtList[ dim ]

    char rcsid[] = "$Id: ls_funcgen.c,v 1.6 1994/05/20 21:49:03 bjax Stab $";

  int       index, prev_index;
  DATA      weight;
  char      *stptr;
  static char *emsg1 = "Normalization value of &f1 less than \n\
lowest breakpoint value &f2 in set &s2.\n";
  static char *emsg2 = "Normalization value of &f1 greater than \n\
largest breakpoint value &f2 in set &s2.\n";
  static char *emsg3 = "Ran off lower end of breakpoint vector &s2 \n\
with normalization value of &f1.\n";
  static char *emsg4 = "Ran off upper end of breakpoint vector &s2 \n\
with normalization value of &f1.\n";

  if (value == nlfunct->latest_bkpt_value[ dim ] )
    return nlfunct->latest_index_and_weights[ dim ];
  if (value < NLFbpl->bkPts[ 0 ] )
    {
      error.severity = warning;
      error.code = E_DATA_INVALID;
      error.strg1 = emsg1;
      error.strg2 = &NLFbpl->name[0];
      error.fp1 = value;
      error.fp2 = NLFbpl->bkPts[ 0 ];
      return 0;
    }
  if (value > NLFbpl->bkPts[ (NLFbpl->length-1) ])
    {
      error.severity = warning;
      error.code = E_DATA_INVALID;
      error.strg1 = emsg2;
      error.strg2 = &NLFbpl->name[0];
      error.fp1 = value;
      error.fp2 = NLFbpl->bkPts[ (NLFbpl->length-1) ];
      return NLFbpl->bkPts[ (NLFbpl->length-1) ];
    }

  /* start looking from last position */

  index = nlfunct->latest_index_and_weights[ dim ];
  if (value < nlfunct->latest_bkpt_value[ dim ] )
    {
      /* search downward */
      prev_index = index + 1;
      while ( NLFbpl->bkPts[ index ] > value )
        {
          prev_index = index;
```

```
        index--;
        if ( index < 0 )
          {
            error.severity = fatal;
            error.code = E_FUNCGEN_INDEX_ERROR;
            error.strg1 = emsg3;
            error.strg2 = &NLFbp1->name[0];
            error.fp1 = value;
            return 0;
          }
      }
    /* found value below - figure weight */

    weight = ( value - NLFbp1->bkPts[ index ] ) /
      (NLFbp1->bkPts[ prev_index ] -
       NLFbp1->bkPts[ index ]);
  }
  else
  {
    /* search upward */
    prev_index = index - 1;
    while ( NLFbp1->bkPts[ index ] < value )
      {
        prev_index = index;
        index++;
        if ( index >= NLFbp1->length )
          {
            error.severity = fatal;
            error.code = E_FUNCGEN_INDEX_ERROR;
            error.strg1 = emsg4;
            error.strg2 = &NLFbp1->name[0];
            error.fp1 = value;
            return 0;
          }
      }
    /* found value below - figure weight */

    weight = ( value - NLFbp1->bkPts[ prev_index ] ) /
      (NLFbp1->bkPts[ index ] - NLFbp1->bkPts[ prev_index ]);
    index = prev_index;
  }
  nlfunct->latest_bkpt_value[ dim ] = value;
  nlfunct->latest_index_and_weights[ dim ] = (DATA)index + weight;
  return nlfunct->latest_index_and_weights[ dim ];
}


DATA getpt( NONLINEAR_FUNCTION *func_ptr, ARG_LIST *arg_list )

{
  int   i, offset, mult;
  DATA  *data_ptr;
  static char *emsg1 = "Function index &i1 value of &i2 exceeds index length \
&i3 in function &s1.\n";

  offset = 0;
  mult = 1;
  for ( i = 0; i < func_ptr->ptr_to_data->dim ; i++ )
    {
      if (arg_list->index[i] > func_ptr->ptr_to_data->length[i])
        {
          error.severity = fatal;
          error.code = E_FUNCGEN_INDEX_ERROR;
          error.strg1 = emsg1;
          strcpy( error.strg2, func_ptr->ptr_to_data->name );
          error.ip1 = i+1;
          error.ip2 = arg_list->index[i]+1;
          error.ip3 = func_ptr->ptr_to_data->length[i]+1;
          return 0;
        }
      offset = offset + (arg_list->index[i])*mult;
      mult = mult * func_ptr->ptr_to_data->length[i];
    }
  data_ptr = (DATA *) func_ptr->ptr_to_data->pts;
  return *(data_ptr+offset);
}

DATA funcgen( NONLINEAR_FUNCTION *func_ptr, ARG_LIST *arg_list, int dim )

{
  DATA                          a, b;
  float                         weight;
  int                                   index;

  if (dim < 0) /* err; */ return 0;
  if (dim > func_ptr->ptr_to_data->dim) /* err; */ return 0;
  arg_list->index[dim] = arg_list->index_and_weight[dim];
  weight = arg_list->index_and_weight[dim] - arg_list->index[dim];

  if (dim == 0)         /* all but first index have been interpreted */
    {
      a = getpt( func_ptr, arg_list );
      arg_list->index[dim]++;
      b = getpt( func_ptr, arg_list );
      arg_list->index[dim]--;
    }
  else  /* more than one dimension to interpret - recurse */
    {
      a = funcgen( func_ptr, arg_list, dim-1);
      arg_list->index[dim]++;
      b = funcgen( func_ptr, arg_list, dim-1);
      arg_list->index[dim]--;
    }
  return (a + weight*(b-a));
}
```

```
/***************************************************************************

        TITLE:  ls_geodesy

----------------------------------------------------------------------------

        FUNCTION:       Converts geocentric coordinates to geodetic positions

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Written as part of LaRCSim project by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    E. B. Jackson

        CODED BY:       E. B. Jackson

        MAINTAINED BY:  E. B. Jackson

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        930208  Modified to avoid singularity near polar region.        EBJ
        930602  Moved backwards calcs here from ls_step.                EBJ
        931214  Changed erroneous Latitude and Altitude variables to
                *lat_geod and *alt in routine ls_geoc_to_geod.          EBJ
        940111  Changed header files from old ls_eom.h style to ls_types,
                and ls_constants.  Also replaced old DATA type with new
                SCALAR type.                                            EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_geodesy.c,v 1.5 1994/01/11 18:47:05 bjax Stab $
$Log: ls_geodesy.c,v $
 * Revision 1.5  1994/01/11  18:47:05  bjax
 * Changed include files to use types and constants, not ls_eom.h
 * Also changed DATA type to SCALAR type.
 *
 * Revision 1.4  1993/12/14  21:06:47  bjax
 * Removed global variable references Altitude and Latitude.   EBJ
 *
 * Revision 1.3  1993/06/02  15:03:40  bjax
 * Made new subroutine for calculating geodetic to geocentric; changed name
 * of forward conversion routine from ls_geodesy to ls_geoc_to_geod.
 *

----------------------------------------------------------------------------

        REFERENCES:

                [ 1]    Stevens, Brian L.; and Lewis, Frank L.: "Aircraft
                        Control and Simulation", Wiley and Sons, 1992.
                        ISBN 0-471-61397-5

----------------------------------------------------------------------------

        CALLED BY:      ls_aux

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:
                lat_geoc        Geocentric latitude, radians, + = North
                radius          C.G. radius to earth center, ft

----------------------------------------------------------------------------

        OUTPUTS:
                lat_geod        Geodetic latitude, radians, + = North
                alt             C.G. altitude above mean sea level, ft
                sea_level_r     radius from earth center to sea level at
                                local vertical (surface normal) of C.G.

--------------------------------------------------------------------------*/

#include "ls_types.h"
#include "ls_constants.h"
#include <math.h>

/* ONE_SECOND is pi/180/60/60, or about 100 feet at earths' equator */
#define ONE_SECOND 4.848136811E-6
#define HALF_PI 0.5*PI


void ls_geoc_to_geod( lat_geoc, radius, lat_geod, alt, sea_level_r )
     SCALAR lat_geoc;
     SCALAR radius;
     SCALAR *lat_geod;
     SCALAR *alt;
     SCALAR *sea_level_r;
{
        SCALAR  t_lat, x_alpha, mu_alpha, delt_mu, r_alpha, l_point, rho_alpha;
        SCALAR  sin_mu_a, denom,delt_lambda, lambda_sl, sin_lambda_sl;

        if(   ( (HALF_PI - lat_geoc) < ONE_SECOND )     /* near North pole */
           || ( (HALF_PI + lat_geoc) < ONE_SECOND ) )   /* near South pole */
          {
            *lat_geod = lat_geoc;
            *sea_level_r = EQUATORIAL_RADIUS*E;
            *alt = radius - *sea_level_r;
          }
        else
          {
            t_lat = tan(lat_geoc);
            x_alpha = E*EQUATORIAL_RADIUS/sqrt(t_lat*t_lat + E*E);
            mu_alpha = atan2(sqrt(RESQ - x_alpha*x_alpha),E*x_alpha);
            if (lat_geoc < 0) mu_alpha = - mu_alpha;
            sin_mu_a = sin(mu_alpha);
            delt_lambda = mu_alpha - lat_geoc;
            r_alpha = x_alpha/cos(lat_geoc);
            l_point = radius - r_alpha;
            *alt = l_point*cos(delt_lambda);
            denom = sqrt(1-EPS*EPS*sin_mu_a*sin_mu_a);
            rho_alpha = EQUATORIAL_RADIUS*(1-EPS)/
              (denom*denom*denom);
            delt_mu = atan2(l_point*sin(delt_lambda),rho_alpha + *alt);
            *lat_geod = mu_alpha - delt_mu;
            lambda_sl = atan( E*E * tan(*lat_geod) ); /* SL geoc. latitude */
```

```
            sin_lambda_sl = sin( lambda_sl );
            *sea_level_r = sqrt(RESQ
                        /(1 + ((1/(E*E))-1)*sin_lambda_sl*sin_lambda_sl));
          }
}

void ls_geod_to_geoc( lat_geod, alt, sl_radius, lat_geoc )
    SCALAR lat_geod;
    SCALAR alt;
    SCALAR *sl_radius;
    SCALAR *lat_geoc;
{
    SCALAR lambda_sl, sin_lambda_sl, cos_lambda_sl, sin_mu, cos_mu, px, py;

    lambda_sl = atan( E*E * tan(lat_geod) ); /* sea level geocentric latitude */
    sin_lambda_sl = sin( lambda_sl );
    cos_lambda_sl = cos( lambda_sl );
    sin_mu = sin(lat_geod);      /* Geodetic (map makers') latitude */
    cos_mu = cos(lat_geod);
    *sl_radius = sqrt(RESQ
        /(1 + ((1/(E*E))-1)*sin_lambda_sl*sin_lambda_sl));
    py = *sl_radius*sin_lambda_sl + alt*sin_mu;
    px = *sl_radius*cos_lambda_sl + alt*cos_mu;
    *lat_geoc = atan2( py, px );
}
```

```
/*************************************************************************

        TITLE:  ls_gravity

--------------------------------------------------------------------------

        FUNCTION:       Gravity model for LaRCsim

--------------------------------------------------------------------------

        MODULE STATUS:  developmental

--------------------------------------------------------------------------

        GENEALOGY:      Created by Bruce Jackson on September 25, 1992.

--------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:  Bruce Jackson

--------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        940111  Changed include files to "ls_types.h" and
                "ls_constants.h" from "ls_eom.h"; also changed DATA types
                to SCALAR types.                                        EBJ

$Header: /aces/larcsim/dev/RCS/ls_gravity.c,v 1.2 1994/01/11 18:50:35 bjax Stab $
$Log: ls_gravity.c,v $
 * Revision 1.2  1994/01/11  18:50:35  bjax
 * Corrected include files (was ls_eom.h) and DATA types changed
 * to SCALARs. EBJ
 *
 * Revision 1.1  1992/12/30  13:18:46  bjax
 * Initial revision
 *

--------------------------------------------------------------------------

                REFERENCES:     Stevens, Brian L.; and Lewis, Frank L.: "Aircraft
                                Control and Simulation", Wiley and Sons, 1992.
                                ISBN 0-471-

--------------------------------------------------------------------------

                CALLED BY:

--------------------------------------------------------------------------

                CALLS TO:

--------------------------------------------------------------------------

                INPUTS:

--------------------------------------------------------------------------

                OUTPUTS:

--------------------------------------------------------------------------*/
#include "ls_types.h"
#include "ls_constants.h"
#include <math.h>

#define GM      1.4076431E16
#define J2      1.08263E-3

void ls_gravity( SCALAR radius, SCALAR lat, SCALAR *gravity )
{

        SCALAR  radius_ratio, rrsq, sinsqlat;

        radius_ratio = radius/EQUATORIAL_RADIUS;
        rrsq = radius_ratio*radius_ratio;
        sinsqlat = sin(lat)*sin(lat);
        *gravity = (GM/(radius*radius))
                *sqrt(2.25*rrsq*rrsq*J2*J2*(5*sinsqlat*sinsqlat -2*sinsqlat + 1)
                                + 3*rrsq*J2*(1 - 3*sinsqlat) + 1);

}
```

```
/***************************************************************************

      TITLE:  ls_ifgl.c

----------------------------------------------------------------------------

      FUNCTION:       Human interface for real-time runs of LaRCSIM
                      models, using Silicon Graphics IRIS workstation.

----------------------------------------------------------------------------

      MODULE STATUS:  Developmental

----------------------------------------------------------------------------

      GENEALOGY:      Created 921230 by Bruce Jackson.

----------------------------------------------------------------------------

      DESIGNED BY:    EBJ

      CODED BY:       EBJ

      MAINTAINED BY:  EBJ

----------------------------------------------------------------------------

      MODIFICATION HISTORY:

      DATE    PURPOSE                                                 BY

      930105  Added help menu capability.                             EBJ
      930111  Added support for passing program name;
              changed "N" to "G" on HUD display.                      EBJ
      930315  Added dummy routine ls_cockpit_exit() for compatibility
              with alternate "curses" cockpit interface.
              Also added dummy routine ls_pause().                    EBJ
      930701  Added bullet model.                                     EBJ
      930802  Added dummy routines for synchronization ls_sync, ls_resync,
              and ls_unsync(), since GL does sync with graphics calls.  EBJ
      930826  Added interface to VMIC 3114 board to read blue
              cockpit stick... sort of kludgey for now.               EBJ
      930921  Benchmarking graphics reveals the following truths for
              the current IRIS ONYX/VTX hardware:

          -   subpixel(TRUE) has little effect
          -   calling swapbuffers() as the last thing in this routine is good
          -   RGBMode is faster than CMAP mode!
          -   v3f() is faster than v3i() !

      931215  Added logic for discrete inputs                         EBJ
      931217  It was inevitable; now we've got a building to blow up.
      931220  Added cockpit structure for passing switch positions
                                                                      EBJ
      931221  Added call to swapinterval() to slow graphics to 20 Hz.
              This was necessary to maintain a steady sim update rate,
              since graphics appear to be raster-fill rate limited,
              and slowdowns are evident when the runway environment fills
              even a WINDOWMARGIN of 80 is specified.                 EBJ
      940105  Renamed this module "ls_ifgl.c" from "ls_glcockpit.c"
              for consistency with other interface routines.  EBJ
      940106  Removed all the old REALSTICK logic for compatibility with
              new sim_control structure; also moved the getwscrn() call to
              after the first winopen() call to try to fix an error when
              running over the network.                               EBJ
      940111  Changed include file from ls_eom.h to ls_types and ls_generic
      940204  Removed ls_sync() and associated dummy routines; real ones
              will be used instead and disabled by control flags if necessary.
      940216  Added hud color variation to signal frame overrun.      EBJ
      940506  Added support for interp. of sim_control_.debug flag    EBJ

<<<<<<< ls_ifgl.c
      940824  Added heading to HUD and centerline to runway.  MLB
      940825  Vsi and navigation information added to HUD.  MLB

      950314  Made VSI use vertical velocity & offsets; HUD now shows global
              variable cockpit_.throttle_pct.                         EBJ
      950316  Increased world size to 400x400 nm; grid spacing set at
              5 nm.                                                   EBJ
      950321  Changed 'A' and 'S' key to drive throttle_pct.          EBJ

$Header: /aces/larcsim/dev/RCS/ls_ifgl.c,v 1.15 1995/03/29 16:11:10 bjax Exp $
=======
$Header: /aces/larcsim/dev/RCS/ls_ifgl.c,v 1.15 1995/03/29 16:11:10 bjax Exp $
>>>>>>> 1.9
$Log: ls_ifgl.c,v $
 * Revision 1.15  1995/03/29  16:11:10  bjax
 * Added calculation to darken sky as we go higher. EBJ
 *
 * Revision 1.14  1995/03/21  13:44:33  bjax
 * Changed use of 'A' and 'S' keys to drive Throttle_pct.
 *
 * Revision 1.13  1995/03/16  13:33:56  bjax
 * Fixed N-S/E-W readouts to nm. EBJ
 *
 * Revision 1.12  1995/03/15  12:18:28  bjax
 * Moved 'paused' variable to sim_control_ common block; reworked
 * pause logic so HUD indicates paused condition more accurately;
 * changed VSI & lat/long readouts to use correct simulation generic
 * variables V_down, D_cg_north_rwy, etc.; changed throttle readout
 * from Throttle[3] to new global variable Throttle_pct; added call
 * to ls_save_current_as_ic() if trim is successful; removed calls
 * to sleep() for GLmouse mode (since sim now starts in paused state).
 * EBJ
 *
 * Revision 1.11  1995/02/27  20:59:46  bjax
 * Added 'T' key that fires off a trim request.
 *
 * Revision 1.9  1994/05/13  17:23:57  bjax
 * Moved around some graphics calls after using gldebug;
 * add'l checks for debug mode prior to calls to swapbuffers();
 *
 * Revision 1.8  1994/05/13  13:16:15  bjax
 * Uncommented-in call to ls_ACES to read stick; this is needed
 * to read the buttons so that a second click of the 'pause' button
 * is read (so that the 'pause' can be cancelled).
 *
 * Revision 1.7  1994/05/10  20:11:48  bjax
 * Commented out call to ls_ACES; this call was moved to main routine
 * to speed up stick reading.  Graphics really should be moved
 * to async process.
 *
 * Revision 1.6  1994/05/06  15:37:30  bjax
 * Removed local "db" flag, and substituted sim_control_.debug flag.
 *
 * Revision 1.5  1994/02/16  12:59:25  bjax
 * Added logic to allow aborts while paused; use HUD to signal overrun.
 *
 * Revision 1.4  1994/02/04  12:59:34  bjax
 * Removed ls_sync() and associated dummy routines.
```

```
 *
 * Revision 1.3  1994/01/11  19:07:55  bjax
 * Fixed include files.
 *
 * Revision 1.2  1994/01/11  18:30:05  bjax
 * Removed REALSTICK macro definition; this duty now performed by
 * sim_control_.sim_type; changed logic for invoking mouse stuff;
 * added explicit return result to getgdesc(GD_TIMERHZ) call.
 *
 * Revision 1.1  1994/01/05  19:57:24  bjax
 * Initial revision
 *
 * Revision 1.9.2.7  1993/12/21  17:40:15  bjax
 * Added call to swapinterval to slow screen update rate to 20 Hz for
 * consistent performance.  This gave time to run with antialiasing and
 * probably z-buffer, although z-buffer causes some problems and isn't worth
 * the effort.
 *
 * Revision 1.9.2.6  1993/12/21  14:34:15  bjax
 * Modified to use new cockpit interface; added speedbrake & gear switches.
 *
 * Revision 1.9.2.4  1993/12/20  16:56:44  bjax
 * Matched the sign convention for target & weapons. EBJ
 *
 * Revision 1.9.2.3  1993/12/17  23:19:24  bjax
 * Building added.
 *
 * Revision 1.9.2.2  1993/12/17  19:15:01  bjax
 * Same version as 1.9.1.13; started new branch.  EBJ
 *
 * Revision 1.9.1.13  1993/12/15  13:52:51  bjax
 * Added support for discrete inputs. EBJ
 *
 * Revision 1.9.1.12  1993/09/23  16:58:48  bjax
 * This version uses multiple polygons to represent the ground, in an
 * (unsuccessful) attempt to allow full-screen 60Hz operation. EBJ
 *
 * Revision 1.9.1.11  1993/09/21  17:01:33  bjax
 * More tuning for graphics: added call to subpixel(), changed almost all
 * calls from v3i() to v3f()'s (bullets are still v3i's); slightly
 * shrunk the window margin size from 100 to 80; increased the gridlines
 * from 9 to 24; moved call to swapbuffers() to the end of the routine (this
 * had a significant improvement in performance). -- EBJ
 *
 * Revision 1.9.1.10  1993/09/17  17:58:11  bjax
 * Okay, lots of changes:
 *    -- Converted most of the ground elements from floating to long int
 *       vertices, in an attempt to avoid loss of 60 Hz sync time
 *    -- Now drawing the ground as a polygon, since circf() call seemed
 *       to take a long time (this might be revisited later, but square ground
 *       looks fine at low altitude)
 *    -- Turned of Gourand shading; switched to flat shading algorithm to
 *       speed things up
 *    -- Went back to non-full screen window, since performance (and 60 Hz
 *       operation) seems to depend EXTREMELY strongly on the number of pixels
 *       in the window.  A margin all around of 100 pixels wide lets the
 *       whole thing run at 60 Hz quite nicely, apparently.
 *    -- This version has the cursor turned off.
 *    -- Added calls to gflush() after each frame is drawn, just to 'make sure'
 *    -- Added call to gexit() when ls_cockpit_exit() is called, just
 *       to 'make sure' we do things right.
 *
 * Revision 1.9.1.9  1993/09/15  18:27:21  bjax
 * This version has stick & throttle read from routine ls_readstick. EBJ
 *
 * Revision 1.9.1.8  1993/09/01  19:27:36  bjax
 * Includes stick interface protocode.
 *
 * Revision 1.9.1.7  1993/08/03  19:04:24  bjax
 * Remember: compile first, then archive!  bjax
 *
 * Revision 1.9.1.6  1993/08/03  19:03:27  bjax
 * Okay, okay, I think I got all the merges.  EBJ
 *
 * Revision 1.9.1.5  1993/08/03  19:02:37  bjax
 * Oops another bug.  EBJ
 *
 * Revision 1.9.1.4  1993/08/03  19:00:34  bjax
 * Finally got the bullets back in , I think.  EBJ
 *
 * Revision 1.9.1.3  1993/08/03  16:33:26  bjax
 * Oops. Fixed problem, I think.  EBJ
 *
 * Revision 1.9.1.2  1993/08/03  16:31:37  bjax
 * Added dummy sync routines.  EBJ
 *
 * Revision 1.9  1993/07/16  19:29:38  bjax
 * Changed RWYLENGTH and RWYWIDTH to floats (added .).  Relocated calls to
 * subpixel(), blendfunction(), and linesmooth() prior to gconfig(), and
 * commented them out after observing performance hit.      EBJ
 *
 * Revision 1.8.1.2  1993/07/02  17:02:06  bjax
 * Got 'em to work.  Bullets now flying! ... and impacting ground.  EBJ
 *
 * Revision 1.8.1.1  93/07/02  13:49:49  bjax
 * This version is intended to have bullets!  Not yet complete.
 *
 * Revision 1.8  93/03/15  08:57:44  bjax
 * Added dummy ls_cockpit_exit() and ls_pause() for compatibility with
 * sun version of ls_main().   EBJ
 *
 * Revision 1.7  93/01/12  08:00:41  bjax
 * Added program name string for window display; changed "N" to "G" on hud.
 *
 * Revision 1.6  93/01/06  09:56:09  bjax
 * Corrected Revision string.
 *
 * Revision 1.5  93/01/06  09:50:24  bjax
 * Added help menu.
 *
 * Revision 1.4  92/12/30  14:15:16  bjax
 * Reversed calls to rotate when DELKEY or PAGEDOWNKEY are depressed.
 *
 * Revision 1.3  92/12/30  14:12:30  bjax
 * Changed call from "init" to "ls_init".
 *
 * Revision 1.2  92/12/30  13:52:21  bjax
 * Changed to point to navion.h in navion directory.
 *
 * Revision 1.1  92/12/30  13:18:18  bjax
 * Initial revision
 *
 * Revision 1.2  93/12/31  10:43:10  bjax
 * Added End, Delete, Page Up/Down keys for view selection.
 *

----------------------------------------------------------------------------

        REFERENCES:
```

```
        The GL world view is oriented +X forward, +Y to left,  and +Z up.

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
/* cockpit.c - performs simple user interface on GL */
#include <gl/gl.h>
#include <gl/device.h>
#include <stdio.h>
#include <math.h>
#include "ls_types.h"
#include "ls_generic.h"
#include "ls_constants.h"
#include "ls_sim_control.h"
#include "ls_cockpit.h"

#define X       0
#define Y       1

#define RGBMODE (comment this line out for colormap mode)

#define WINDOWMARGIN 0

#ifdef GLPROF
#define OBJ(x) glprof_object(x);
#else
#define OBJ(x)
#endif

/* ground grid. MAXGRID is +/- ft from rwy threshold */
#define MAXGRID 200.*6076.
#define GRIDLINES 40    /* 5 nm grids */

#define ATMOS_MAX 420000.       /* max altitude of blue sky */

/* stick (mouse) and discrete gearing */
#define LON_SCALE 0.3
#define LAT_SCALE 0.3
#define DELTAARROWAMT 20
#define DELTARUDDER 0.01

/* top-to-bottom viewing angle */
#define YWINDOWANGLE 30.

/* Head-up Display geometry: X is +left, Y is + up */
#define HUDDIST 200
#define HUDBORESIGHTSIZE 1
#define HUDLADDERWIDTH 20
#define HUDLADDERWINGLETLENGTH 2
#define HUDVELX 12
#define HUDVELY  0
#define HUDALTX -4
#define HUDALTY  0
#define HUDDATAX 30
#define HUDMACHY   -4
#define HUDALPHAY  -6
#define HUDNZY     -8
#define HUDTY     -10

/* runway geometry */
#define RWYLENGTH 15000.
#define RWYWIDTH 300.

/* Name of program that invoked this application */
extern char *progname;

extern SCALAR Simtime;

/* variables with FILE visibility */
static Matrix mhome;
static int trigger = 0;

typedef struct
    {
        double x, y, z, killradius2;
        int hit;
    } target,  *ptarget;

static int targets_alive = 0;

#define MAXTARGETS 10

static ptarget targetlist[MAXTARGETS];

/* cockpit interface data block  - defn's in ls_cockpit.h */

COCKPIT cockpit_;


void drawsky()
{

#define SKYCOLOR 3*64-13
  static short skycolor0[3]    = (  63,  63, 255 );
  short skycolor[3];
  int i;
  float factor;


  OBJ("drawsky"); /* marker for GLProf */

  /* to provide sky darkening - goes to black at ATMOS_MAX */

  factor = Altitude/ATMOS_MAX;
  if (factor < 0.) factor = 0.0;
  if (factor > 1.) factor = 1.0;
  factor = sqrt( 1.0 - pow(factor,2) );
  for(i=0;i<3;i++) skycolor[i] = factor*skycolor0[i];


#ifdef RGBMODE
  c3s(skycolor);
#else
  color(SKYCOLOR);
#endif
```

```c
  clear();
}


void drawground()
{

#define GROUNDCOLOR 59
  static short groundcolor[3] = {   8, 127,   8 };
#define GROUNDSIZE MAXGRID
  int i, j;
  float vert0[3], vert1[3];
  float grdspace;
  float d;

  OBJ("drawground");  /* marker for GLProf */

#ifdef RGBMODE
  c3s(groundcolor);
#else
  color(GROUNDCOLOR);
#endif
  grdspace = MAXGRID/GRIDLINES;

  vert0[2] = Runway_altitude;
  vert1[2] = Runway_altitude;

  /* outer loop - south to north progression of quadrilateral strips */
  for(i=0; i<2*GRIDLINES; i++)
  {
    bgnqstrip();

      vert0[1] = -MAXGRID+(i*grdspace);  /* southern edge coordinate */
      vert1[1] = vert0[1] + grdspace;    /* northern edge coordinate */
      /* inner loop - east to west quadstrips */
      for(j=2*GRIDLINES; j>=0; j--)
      {
          vert0[0] = -MAXGRID+(j*grdspace);
          vert1[0] = vert0[0];
          v3f(vert0);
          v3f(vert1);
      }

    endqstrip();
  }
}


void drawgrid()
{
#define GRIDCOLOR BLACK
  static short gridcolor[3]   = {   0,    0,    0 };
  static float grid[5][3] = {
    { -MAXGRID, -MAXGRID, 0. },
    {  MAXGRID, -MAXGRID, 0. },
    {  MAXGRID,  MAXGRID, 0. },
    { -MAXGRID,  MAXGRID, 0. },
    { -MAXGRID, -MAXGRID, 0. }
  };
  static inited = 0;
  static float grdspace;

  int i;
  float d;
  float vert[3];

  OBJ("drawgrid");  /* marker for GLProf */

  if (!inited)
  {
    inited = -1;
    grdspace = MAXGRID/GRIDLINES;
    grid[0][2] = Runway_altitude+0.25;
    grid[1][2] = Runway_altitude+0.25;
    grid[2][2] = Runway_altitude+0.25;
    grid[3][2] = Runway_altitude+0.25;
    grid[4][2] = Runway_altitude+0.25;
  }

  /* add gridlines */

#ifdef RGBMODE
  c3s(gridcolor);
#else
  color(GRIDCOLOR);
#endif


  /* outline playing field */
  bgnline();
    for(i=0;i<5;i++) v3f(grid[i]);
  endline();

  /* draw gridlines */
  vert[2] = Runway_altitude;
  for(i=1; i<2*GRIDLINES; i++)
  {
    d = -MAXGRID+(i*grdspace);

    /* draw N-S gridlines */
    vert[0] = d;
    bgnline();
      vert[1] = -MAXGRID;  v3f( vert );
      vert[1] =  MAXGRID;  v3f( vert );
    endline();

    /* draw E-W gridlines */
    vert[1] = d;
    bgnline();
      vert[0] = -MAXGRID;  v3f( vert );
      vert[0] =  MAXGRID;  v3f( vert );
    endline();
  }
}


void drawrwy()
{
#define RWYCOLOR 8
  static short rwycolor[3]    = { 127, 127, 127 };
  static float vert1_track[3],vert2_track[3];
  static short linecolor[3] = { 0, 0, 0 };
  static float rwy[5][3] = {
    { 0,          RWYWIDTH/2, 0. },
    { RWYLENGTH,  RWYWIDTH/2, 0. },
    { RWYLENGTH, -RWYWIDTH/2, 0. },
    { 0,         -RWYWIDTH/2, 0. },
```

```
    { 0,          RWYWIDTH/2, 0. }
  );
  static inited = 0;

  int i;

  OBJ("drawrwy");  /* marker for GLProf */

    if(!inited)
    {
        inited = -1;
        rwy[0][2] = Runway_altitude+0.5;
        rwy[1][2] = Runway_altitude+0.5;
        rwy[2][2] = Runway_altitude+0.5;
        rwy[3][2] = Runway_altitude+0.5;
        rwy[4][2] = Runway_altitude+0.5;
    }

#ifdef RGBMODE
  c3s(rwycolor);
#else
  color(RWYCOLOR);
#endif

  /* add a runway */
  bgnpolygon();
  for(i=0;i<5;i++)
    v3f(rwy[i]);
  endpolygon();

  /* outline rwy */
#ifdef RGBMODE
  cpack(0x00000000);/* black */
#else
  color(BLACK);
#endif
  bgnline();
  for(i=0;i<5;i++)
    v3f(rwy[i]);
  endline();

/* rwy centerline */
vert1_track[0] = 1000.0;
vert1_track[1] = 0.0;
vert1_track[2] = 0.0;
vert2_track[0] = 1500.0;
vert2_track[1] = 0.0;
vert2_track[2] = 0.0;

/* draw rwy centerline */
/* deflinestyle(1,0xFFFD);*/
/* setlinestyle(1);*/
linewidth(3);
if(Altitude > 50.) linewidth(2);
if(Altitude > 500.)linewidth(1);
c3s(linecolor);
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 2000.0;
vert2_track[0] = 2500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 3000.0;
vert2_track[0] = 3500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 4000.0;
vert2_track[0] = 4500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 5000.0;
vert2_track[0] = 5500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 6000.0;
vert2_track[0] = 6500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 7000.0;
vert2_track[0] = 7500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 8000.0;
vert2_track[0] = 8500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 9000.0;
vert2_track[0] = 9500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 10000.0;
vert2_track[0] = 10500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 11000.0;
vert2_track[0] = 11500.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
```

```
endline();

vert1_track[0] = 12000.0;
vert2_track[0] = 12750.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();

vert1_track[0] = 13250.0;
vert2_track[0] = 14000.0;
bgnline();
v3f(vert1_track);
v3f(vert2_track);
endline();


linewidth(1);
/*setlinestyle(0);*/
}

void drawtargets()
{
#define TARGCOLOR 3
#define TARGSIZE 20.
#define TARG_X_IC 7500.
#define TARG_Y_IC 500.
#define TARG_Z_IC 0.
  static short targcolor[3]    = { 30, 30, 20 };
  static struct {
    int exists;
    double x, y, z;
    double xdot, ydot, zdot;
    double p, q, r;
    double phi, theta, psi;
    } piece[5];

  static float targ_geom[5][5][3] =
  {
      { /* east face */
        {  TARGSIZE, -TARGSIZE,  0.},
        {  TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE, -TARGSIZE,  0.},
        {  TARGSIZE, -TARGSIZE,  0.},
      },
      { /* north face */
        {  TARGSIZE,  TARGSIZE,  0.},
        {  TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE, -TARGSIZE,  0.},
        {  TARGSIZE,  TARGSIZE,  0.},
      },
      { /* west face */
        { -TARGSIZE,  TARGSIZE,  0.},
        { -TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE,  TARGSIZE,  0.},
        { -TARGSIZE,  TARGSIZE,  0.},
      },
      { /* south face */
        { -TARGSIZE, -TARGSIZE,  0.},
        { -TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE,  TARGSIZE,  0.},
        { -TARGSIZE, -TARGSIZE,  0.},
      },
      { /* top face */
        {  TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE,  TARGSIZE,  2.0*TARGSIZE },
        { -TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
        {  TARGSIZE, -TARGSIZE,  2.0*TARGSIZE },
      }
  };
  static inited = 0;
  static hit_init = 0;
  static double oldSimtime = 5.;
  static double targ_x, targ_y, targ_z;
  double dt;
  int i, j;

  OBJ("drawtargets");  /* marker for GLProf */

    if(!inited)
    {
        inited = -1;
        targ_x = TARG_X_IC;
        targ_y = TARG_Y_IC;
        targ_z = TARG_Z_IC;
        targetlist[0] = (ptarget) malloc(sizeof(target));
        if (targetlist[0] == 0L) return;
        targetlist[0]->x = targ_x;
        targetlist[0]->y = targ_y;
        targetlist[0]->z = targ_z+TARGSIZE;
        targetlist[0]->killradius2=4*TARGSIZE*TARGSIZE;
        targetlist[0]->hit = 0;
    }
    if (oldSimtime > Simtime)
        {
            targets_alive = 1;
            targetlist[0]->hit = 0;
            oldSimtime = Simtime;
            hit_init = 0;
            for(i = 0; i<5; i++)
            {
                piece[i].exists = 0;
                piece[i].x = targ_x;
                piece[i].y = targ_y;
                piece[i].z = targ_z;
                piece[i].xdot = 0.;
                piece[i].ydot = 0.;
                piece[i].zdot = 0.;
                piece[i].p = 0.;
                piece[i].q = 0.;
                piece[i].r = 0.;
                piece[i].phi = 0.0;
                piece[i].theta = 0.0;
                piece[i].psi = 0.0;
            }
        }

    dt = Simtime - oldSimtime;
    oldSimtime = Simtime;


#ifdef RGBMODE
  c3s(targcolor);
#else
  color(TARGCOLOR);
```

```
#endif

  /* draw the target */
  if (!targetlist[0]->hit)
  {
    pushmatrix();
    translate( targ_x, -targ_y, targ_z );
    for(i = 0; i<5; i++)
    {
        bgnpolygon();
        for(j = 0; j<5; j++)
        {
            v3f(targ_geom[i][j]);
        }
        endpolygon();
    }
    popmatrix();
  }
  else
  {
      if(!hit_init)
      {
          hit_init=-1;

          targets_alive--;

          piece[0].exists = 1;
          piece[0].zdot = 100.;
          piece[0].xdot = 0.;
          piece[0].ydot = -300.;
          piece[0].p = 20.;
          piece[0].q = -10.;
          piece[0].r = 300.;

          piece[1].exists = 1;
          piece[1].zdot = 50.;
          piece[1].xdot = 300.;
          piece[1].ydot = 10.;
          piece[1].p = 200.;
          piece[1].q = -20.;
          piece[1].r = 2000.;

          piece[2].exists = 1;
          piece[2].zdot = 200.;
          piece[2].xdot = 0.;
          piece[2].ydot = 300.;
          piece[2].p = -50.;
          piece[2].q = 20.;
          piece[2].r = 5000.;

          piece[3].exists = 1;
          piece[3].zdot = 100.;
          piece[3].xdot = -300.;
          piece[3].ydot = 10.;
          piece[3].p = -200.;
          piece[3].q = 20.;
          piece[3].r = 200.;

          piece[4].exists = 1;
          piece[4].zdot = 100.;
          piece[4].xdot = -3.;
          piece[4].ydot = 10.;
          piece[4].p = 200.;
          piece[4].q = 20.;
          piece[4].r = 2000.;
      }
      for(i = 0; i<5; i++)
      {
          if (piece[i].exists)
            {
                piece[i].zdot  = piece[i].zdot  - dt*Gravity;
                piece[i].x     = piece[i].x     + dt*piece[i].xdot;
                piece[i].y     = piece[i].y     + dt*piece[i].ydot;
                piece[i].z     = piece[i].z     + dt*piece[i].zdot;
                piece[i].phi   = piece[i].phi   + dt*piece[i].p;
                piece[i].theta = piece[i].theta + dt*piece[i].q;
                piece[i].psi   = piece[i].psi   + dt*piece[i].r;
                if(piece[i].z < Runway_altitude) piece[i].exists = 0;
                pushmatrix();
                translate( piece[i].x, -piece[i].y, piece[i].z );
                rotate( (int) piece[i].phi, 'x');
                rotate( (int) piece[i].theta, 'y');
                rotate( (int) piece[i].psi, 'z');
                bgnpolygon();
                for(j = 0; j<5; j++)
                {
                    v3f(targ_geom[i][j]);
                }
                endpolygon();
                popmatrix();
            }
      }
  }
}


void drawhud( merect )

  Matrix merect;     /* erect matrix centered at eyepoint */

{
#define HUDCOLOR WHITE
  static short hudcolor[3]    = { 255, 255, 255 };
  static short slocolor[3]    = { 255, 0, 0 };
  static float hudboresight[4][3] = {
    { HUDDIST, -HUDBORESIGHTSIZE,  0 },
    { HUDDIST,  HUDBORESIGHTSIZE,  0 },
    { HUDDIST,  0, -HUDBORESIGHTSIZE },
    { HUDDIST,  0,  HUDBORESIGHTSIZE }
  };
  static float vvect[6][3] = {      /* velocity vector wing & tail */
    {   -HUDBORESIGHTSIZE,  0,  0 },
    { -2*HUDBORESIGHTSIZE,  0,  0 },
    {  0,    HUDBORESIGHTSIZE,  0 },
    {  0,  2*HUDBORESIGHTSIZE,  0 },
    {  0,   -HUDBORESIGHTSIZE,  0 },
    {  0, -2*HUDBORESIGHTSIZE,  0 }
  };
  static float pvect[6][3] = {      /* pitch ladder */
    { HUDDIST,  0.5*HUDLADDERWIDTH, 0},
    { HUDDIST,  0.5*HUDLADDERWIDTH, 0},
    { HUDDIST,  3*HUDBORESIGHTSIZE, 0},
    { HUDDIST, -0.5*HUDLADDERWIDTH, 0},
    { HUDDIST, -0.5*HUDLADDERWIDTH, 0},
    { HUDDIST, -3*HUDBORESIGHTSIZE, 0}
  };
  static float nvect[4][3] = {      /* nadir marker */
    {  HUDBORESIGHTSIZE,  HUDBORESIGHTSIZE, -HUDDIST },
    { -HUDBORESIGHTSIZE, -HUDBORESIGHTSIZE, -HUDDIST },
```

```
    { HUDBORESIGHTSIZE, -HUDBORESIGHTSIZE, -HUDDIST },
    { -HUDBORESIGHTSIZE,  HUDBORESIGHTSIZE, -HUDDIST }
  };
  char *altstr="999999.9";
  char *velstr="999999.9";
  char *machstr= "M 9.99";
  char *alphstr= "A 99.9";
  char *nzstr  = "G 99.9";
  char *thrstr = "T 20%%";
  char *Psistr = "999.9";
  char *Vsistr="999999.9";
  char *Latstr="999999.9";
  char *Lonstr="999999.9";
  float ang, angrad;
  static float tyme;
  static float tymep;
  static float vsi;
  static float vsip;
  static float altp;


  OBJ("drawhud");  /* marker for GLProf */

  pushmatrix();   /* save hud centered matrix */

  /* note: at this point, screen coordinates are
     +X away from eye, +Y to left, and +Z up     */

#ifdef RGBMODE
  c3s(hudcolor);
#else
  color(HUDCOLOR);
#endif

  if( sim_control_.overrun || sim_control_.paused ) c3s(slocolor); /* signal less than r
eal-time */
  /* draw bore sight */
  bgnline();
  {
    v3f(hudboresight[0]);
    v3f(hudboresight[1]);
  }
  endline();
  bgnline();
  {
    v3f(hudboresight[2]);
    v3f(hudboresight[3]);
  }
  endline();

  /* write alphanumeric data */
  cmovi(HUDDIST, HUDVELX, HUDVELY);
  sprintf(velstr, "%5.0f", V_equiv_kts);
  charstr(velstr);

  cmovi(HUDDIST, HUDALTX, HUDALTY);
  sprintf(altstr, "%5.0f", Altitude);
  charstr(altstr);

  cmovi(HUDDIST, HUDDATAX, HUDMACHY);
  sprintf(machstr, "M %3.2f", Mach_number);
  charstr(machstr);

  cmovi(HUDDIST, HUDDATAX, HUDALPHAY);
  sprintf(alphstr, "A %3.1f", Alpha*57.3);
```



```
  charstr(alphstr);

  cmovi(HUDDIST, HUDDATAX, HUDNZY);
  sprintf(nzstr, "G %3.2f", -N_Z_cg);
  charstr(nzstr);

  cmovi(HUDDIST, HUDDATAX, HUDTY);
  sprintf(thrstr, "T %3.0f%%", Throttle_pct*100.);
  charstr(thrstr);

  cmovi(HUDDIST, 1.5, 5.); /* add heading info to HUD -jbd*/
  sprintf(Psistr, "%2.0f", Psi*57.3);
  charstr(Psistr);

  cmovi(HUDDIST, -6., -4.); /* add vertical speed to HUD -mlb*/
  sprintf(Vsistr, "%5.0f", -V_down*60.);
  charstr(Vsistr);

  cmovi(HUDDIST, -20., -8.); /* add navigation to HUD -mlb*/
  sprintf(Latstr, "N-S %5.2f", D_pilot_north_of_rwy/6076.);
  charstr(Latstr);

  cmovi(HUDDIST, -20., -10.); /* add navigation to HUD -mlb*/
  sprintf(Lonstr, "E-W %5.2f", D_pilot_east_of_rwy/6076.);
  charstr(Lonstr);

  /* draw velocity vector */

  rot(-Beta*RAD_TO_DEG, 'z');
  rot(Alpha*RAD_TO_DEG, 'y');
  translate( HUDDIST, 0, 0);
  rotate(900, 'y');

  arc(0, 0, HUDBORESIGHTSIZE, 1, 3600);
  bgnline();
     v3f( vvect[0] );
     v3f( vvect[1] );
  endline();
  bgnline();
     v3f( vvect[2] );
     v3f( vvect[3] );
  endline();
  bgnline();
     v3f( vvect[4] );
     v3f( vvect[5] );
  endline(); ,


  /* draw pitch ladder */

  loadmatrix(merect);    /* set up for eyepoint centered erect drawing */
  rot(-Psi*RAD_TO_DEG, 'z');

  /* draw horizon line */
  pvect[0][1] = HUDLADDERWIDTH;
  pvect[0][2] = 0;
  pvect[3][1] = -pvect[0][1];
  pvect[3][2] = 0;
  bgnline();
    v3f( pvect[0] );
    v3f( pvect[2] );
  endline();
  bgnline();
    v3f( pvect[3] );
    v3f( pvect[5] );
```

```
endline();

/* draw apex marker */
pushmatrix();         /* save pitch ladder coordinates */
translate(0, 0, +HUDDIST);
arc(0, 0, HUDBORESIGHTSIZE, 1, 3600);
popmatrix();

/* draw nadir marker */
bgnline();
  v3f(nvect[0]);
  v3f(nvect[1]);
endline();
bgnline();
  v3f(nvect[2]);
  v3f(nvect[3]);
endline();

/* draw upper pitch ladder */
pushmatrix();         /* save pitch ladder coordinates */
for(ang=5.; ang < 85.1; ang=ang+5)
  {
    rotate(-50, 'y');
    angrad = ang*DEG_TO_RAD;
    pvect[0][1] = 0.5*HUDLADDERWIDTH + HUDLADDERWINGLETLENGTH*cos(angrad);
    pvect[0][2] = -HUDLADDERWINGLETLENGTH*sin(angrad);
    pvect[3][1] = -pvect[0][1];
    pvect[3][2] = pvect[0][2];
    bgnline();
      v3f( pvect[0] );
      v3f( pvect[1] );
      v3f( pvect[2] );
    endline();
    bgnline();
      v3f( pvect[3] );
      v3f( pvect[4] );
      v3f( pvect[5] );
    endline();
  }
popmatrix();  /* restore pitch ladder coordinates */

/* draw "backside" 15 degrees beyond apex */
pushmatrix();         /* save pitch ladder coordinates */
rotate(-900, 'y');
for(ang=5.; ang < 15.1; ang=ang+5)
  {
    rotate(-50, 'y');
    angrad = (90 - ang)*DEG_TO_RAD;
    pvect[0][1] = 0.5*HUDLADDERWIDTH + HUDLADDERWINGLETLENGTH*cos(angrad);
    pvect[0][2] = HUDLADDERWINGLETLENGTH*sin(angrad);
    pvect[3][1] = -pvect[0][1];
    pvect[3][2] = pvect[0][2];
    bgnline();
      v3f( pvect[0] );
      v3f( pvect[1] );
      v3f( pvect[2] );
    endline();
    bgnline();
      v3f( pvect[3] );
      v3f( pvect[4] );
      v3f( pvect[5] );
    endline();
  }
popmatrix();  /* restore pitch ladder coordinates */

/* draw lower pitch ladder */
pushmatrix();   /* save pitch ladder coordinates */
for(ang=-5.; ang > -85.1; ang=ang-5)
  {
    rotate( 50, 'y');
    angrad = ang*DEG_TO_RAD;
    pvect[0][1] = 0.5*HUDLADDERWIDTH + HUDLADDERWINGLETLENGTH*cos(angrad);
    pvect[0][2] = -HUDLADDERWINGLETLENGTH*sin(angrad);
    pvect[3][1] = -pvect[0][1];
    pvect[3][2] = pvect[0][2];
    bgnline();
      v3f( pvect[0] );
      v3f( pvect[1] );
      v3f( pvect[2] );
    endline();
    bgnline();
      v3f( pvect[3] );
      v3f( pvect[4] );
      v3f( pvect[5] );
    endline();
  }
popmatrix();  /* restore pitch ladder coordinates matrix */


/* draw "backside" 15 degrees beyond nadir */
pushmatrix();         /* save pitch ladder coordinates */
rotate(900, 'y');
for(ang=-5.; ang > -15.1; ang=ang-5)
  {
    rotate( 50, 'y');
    angrad = (-90 - ang)*DEG_TO_RAD;
    pvect[0][1] = 0.5*HUDLADDERWIDTH + HUDLADDERWINGLETLENGTH*cos(angrad);
    pvect[0][2] = HUDLADDERWINGLETLENGTH*sin(angrad);
    pvect[3][1] = -pvect[0][1];
    pvect[3][2] = pvect[0][2];
    bgnline();
      v3f( pvect[0] );
      v3f( pvect[1] );
      v3f( pvect[2] );
    endline();
    bgnline();
      v3f( pvect[3] );
      v3f( pvect[4] );
      v3f( pvect[5] );
    endline();
  }
popmatrix();  /* restore pitch ladder coordinates */

popmatrix();  /* restore HUD centered coordinates */

}


void drawweapons()
{
#define MAXBULLETS 50
#define FIREINTERVAL 0.10
#define LIFETIME 20.
#define MUZZLEVEL 2000.
#define BULLETSIZE 1
#define BULLETCOLOR 1
#define EXPLOSIONCOLOR 7
    static short bulletcolor[3] = {  255, 0,  0 };
    static short explosioncolor[3] = {  255, 255,  255 };
```

```
    static float bvect[4][3] = { /* bullet image */
        {  BULLETSIZE,  0,  0 },
        {  0, 0, BULLETSIZE   },
        { -BULLETSIZE,  0,  0 },
        {  BULLETSIZE,  0,  0 }
    };
    static float expl[4][3] = { /* explosion image */
        {  10*BULLETSIZE,  0,  0 },
        {  0, 0, 10*BULLETSIZE   },
        { -BULLETSIZE*10,  0,  0 },
        {  BULLETSIZE*10,  0,  0 }
    };


    typedef struct
        {
            double age, xdot, ydot, zdot, x, y, z;
        } bullet,  *pbullet;

    static  int bullets_away = 0;

    static pbullet bulletlist[MAXBULLETS];

    static double oldSimtime = 5.;
    static double lastFiredtime = 0.;
    double x_miss, y_miss, z_miss, miss2;
    double dt;
    int i, j;

    OBJ("drawweapons");  /* marker for GLProf */

    if (oldSimtime > Simtime)
        {
            for (i = 0; i < bullets_away; i++)
                free( bulletlist[i] );
            bullets_away = 0;
            oldSimtime = Simtime;
            lastFiredtime = 0.;
        }

    dt = Simtime - oldSimtime;
    oldSimtime = Simtime;

    if (bullets_away || trigger)
        {
          if (trigger && (bullets_away < MAXBULLETS)
            && (Simtime > lastFiredtime + FIREINTERVAL))
            {
                lastFiredtime = Simtime;
                bulletlist[bullets_away] = (pbullet) malloc(sizeof(bullet));
                if (bulletlist[bullets_away] == 0L) return;
                bulletlist[bullets_away]->age = 0.;
                bulletlist[bullets_away]->xdot =
                    V_north_rel_ground + MUZZLEVEL*T_local_to_body_11;
                bulletlist[bullets_away]->ydot =
                    V_east_rel_ground  + MUZZLEVEL*T_local_to_body_12;
                bulletlist[bullets_away]->zdot =
                    -V_down_rel_ground  - MUZZLEVEL*T_local_to_body_13;
                bulletlist[bullets_away]->x = X_cg_rwy;
                bulletlist[bullets_away]->y = Y_cg_rwy;
                bulletlist[bullets_away]->z = H_cg_rwy;

                bullets_away++;

            }
          trigger = 0; /* to clear the trigger even if we were out of bullets */
          for (i = 0; i < bullets_away; i++)
            {
                bulletlist[i]->age = bulletlist[i]->age + dt;
                if (bulletlist[i]->age > LIFETIME)
                    {
                        free (bulletlist[i]);
                        for (j = i; j < (bullets_away-1); j++)
                            bulletlist[j] = bulletlist[j+1];
                        bullets_away--;
                        bulletlist[bullets_away] = 0L;
                    }
                else
                    {
                        bulletlist[i]->zdot = bulletlist[i]->zdot - Gravity*dt;
                        bulletlist[i]->x = bulletlist[i]->x + bulletlist[i]->xdot*dt
;
                        bulletlist[i]->y = bulletlist[i]->y + bulletlist[i]->ydot*dt
;
                        bulletlist[i]->z = bulletlist[i]->z + bulletlist[i]->zdot*dt
;
                    }
            }
          for (i = 0; i < bullets_away; i++)
            {
              pushmatrix();
              translate(bulletlist[i]->x, -bulletlist[i]->y, bulletlist[i]->z);
              rotate( (int) (-(Psi*57.3+90.)*10.), 'Z');
              rotate( (int) (Theta*573.),  'X');
              #ifdef RGBMODE
                c3s(bulletcolor);
              #else
                color(BULLETCOLOR);
              #endif
              bgnpolygon();
                for(j = 0; j < 4; j++) v3f(bvect[j]);
              endpolygon();
              if(bulletlist[i]->z < 0)
                {
                  #ifdef RGBMODE
                    c3s(explosioncolor);
                  #else
                    color(EXPLOSIONCOLOR);
                  #endif
                  bgnpolygon();
                    for(j = 0; j < 4; j++) v3f(expl[j]);
                  endpolygon();
                  bulletlist[i]->age = Simtime + LIFETIME;
                }
              popmatrix();

              /* check for hits on each target */

              for(j = 0; j < targets_alive; j++)
                {
                  x_miss = bulletlist[i]->x - targetlist[j]->x;
                  y_miss = bulletlist[i]->y - targetlist[j]->y;
                  z_miss = bulletlist[i]->z - targetlist[j]->z;
                  miss2 = x_miss*x_miss + y_miss*y_miss + z_miss*z_miss;
                  if (miss2 < targetlist[j]->killradius2)
                    targetlist[j]->hit = -1;
                }
            }
        }
    return;
```

```
}


void drawworld(phi, theta, psi, xrwy, yrwy, alt, hudon)

float phi, theta, psi, xrwy, yrwy, alt;
int hudon;
{
  Matrix merect;

  OBJ("drawworld");  /* marker for GLProf */

  if (hudon) pushmatrix();     /* save current eyepoint for HUD */

  rot(-phi,   'x');
  rot( theta, 'y');
  rot( psi,   'z');

  getmatrix(merect);    /* save rotated matrix for HUD */

  translate(-xrwy, yrwy, -alt);

  drawsky();
  drawground();
  drawgrid();
  drawrwy();
  drawtargets();
  drawweapons();

  if (hudon)
    {
      popmatrix();
      drawhud(merect);
    };


}

void ls_help()
{

#define HELPBKGND 8
#define HELPCOLOR WHITE
  static short helpbkgnd[3]  = { 127, 127, 127 };
  static short helpcolor[3]  = { 255, 255, 255 };

  int row;
  Device dev;
  short val;

  ls_unsync();  /* disable timer interrupts */

#ifdef RGBMODE
  c3s(helpbkgnd);
#else
  color(HELPBKGND);
#endif

  pushmatrix(); /* save the current drawing frame matrix */
  clear();      /* write the background color */

  loadmatrix( mhome );

#ifdef RGBMODE
    c3s(helpcolor);
#else
    color(HELPCOLOR);
#endif

/* Note: in this frame, +X is left, +Y is up */
#define HELPDIST HUDDIST
#define HELPXLEFT +20
#define HELPXRIGHT 0
#define HELPROWINC -4
#define HELPSTARTROW +40

  cmovi( HELPDIST, HELPXLEFT, HELPSTARTROW );
  charstr("LaRCSIM HELP MENU");
  cmovi( HELPDIST, HELPXLEFT, HELPSTARTROW + HELPROWINC/2);
  charstr("EBJ/AGCB/GCD/FSD/LaRC/NASA");
  cmovi( HELPDIST, HELPXLEFT, HELPSTARTROW + HELPROWINC );
  charstr("$Date: 1995/03/29 16:11:10 $");

  row = HELPSTARTROW + 3*HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("ESC");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Quit");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("'?'");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Help");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("'a'");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Retard throttle");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("'s'");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Advance throttle");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("Mouse buttons");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Left, center, right rudder");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("'r'");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Reset sim");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("'p'");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Pause sim (second 'p' to restart)");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("Arrow keys");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Look around");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("Home");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Forward view");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("End");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Rear view");
  row = row + HELPROWINC;

  cmovi( HELPDIST, HELPXLEFT, row  );charstr("Insert");
  cmovi( HELPDIST, HELPXRIGHT, row );charstr("Left view");
  row = row + HELPROWINC;
```

```
    cmovi( HELPDIST, HELPXLEFT, row  );charstr("Page Up");
    cmovi( HELPDIST, HELPXRIGHT, row );charstr("Right view");
    row = row + HELPROWINC;

    cmovi( HELPDIST, HELPXLEFT, row  );charstr("Delete");
    cmovi( HELPDIST, HELPXRIGHT, row );charstr("Left downward view");
    row = row + HELPROWINC;

    cmovi( HELPDIST, HELPXLEFT, row  );charstr("Page Down");
    cmovi( HELPDIST, HELPXRIGHT, row );charstr("Right downward view");
    row = row + 2*HELPROWINC;


    cmovi( HELPDIST, HELPXLEFT, row );charstr("Press any key to return to flight...");


    if(!sim_control_.debug) swapbuffers();

    qdevice(KEYBD);
    qreset();
    dev = qread(&val);    /* wait for user input */
    unqdevice(KEYBD);
    qreset();

    popmatrix();     /* reload original drawing matrix */

    ls_resync();

  }
```



```
  int ls_cockpit( )

  {
    int abort,  db;

    static int old_left_but, old_right_but, old_first_trig, old_second_trig;

    static short val, mval[2], mbias[2];
    static long org[2], size[2], win;
    long xWindSize,  yWindSize;
    static Device dev, mdev[2];
    static double fscale[2];

    static int inited = FALSE;
    static int hudon = TRUE;

    static unsigned short cros[16] = {
      0x0100, 0x0100, 0x0100, 0x0100,
      0x0100, 0x0100, 0x0100, 0xfffe,
      0x0100, 0x0100, 0x0100, 0x0100,
      0x0100, 0x0100, 0x0100, 0x0000
      };

    float ar;

    long savescrn, gd_timerhz;
    static short blk[3] = {0, 0, 0};

    glprof_object("ls_cockpit");  /* marker for GLProf */

    if (!inited)
      {
        gd_timerhz = getgdesc(GD_TIMERHZ);

        /* create window */
        xWindSize = getgdesc(GD_XPMAX) - 2*WINDOWMARGIN;
        yWindSize = getgdesc(GD_YPMAX) - 2*WINDOWMARGIN;
        prefposition(WINDOWMARGIN, xWindSize + WINDOWMARGIN,
                     WINDOWMARGIN, yWindSize + WINDOWMARGIN);
        foreground();
        winopen(progname);
        mmode(MVIEWING);
        savescrn = scrnselect(getwscrn());
        scrnselect(savescrn);
        blanktime( 1800*gd_timerhz ); /* delay timeout for half hour */
        if(!sim_control_.debug)
          {
            doublebuffer();  /* so it can be overridden in debug mode */
          }
        shademodel(FLAT);
#ifdef RGBMODE
        RGBmode();
#endif
        subpixel(TRUE);

        gconfig();

        if(!sim_control_.debug)
          {
            swapinterval( ( short ) 3 );    /* should force sim to run at 20 Hz */
          }

        blendfunction(BF_SA, BF_MSA);
        linesmooth(SML_SMOOTHER);

        ar = (float)xWindSize/(float)yWindSize;

        perspective(YWINDOWANGLE*10, ar, 10., 1000000.);
        polarview(0., -900, 900, 0);
#ifdef RGBMODE
        c3s(blk);
#else
        color(BLACK);
#endif
        if(!sim_control_.debug) swapbuffers();

        getmatrix( mhome );

        pushmatrix();
        drawworld(Phi*RAD_TO_DEG, Theta*RAD_TO_DEG, Psi*RAD_TO_DEG,
                  X_pilot_rwy, Y_pilot_rwy, H_pilot_rwy, hudon );
        if(!sim_control_.debug) swapbuffers();
        popmatrix();


        /* set up to read keys */
        qdevice(SPACEKEY);
        qdevice(ESCKEY);
        qdevice(AKEY);
        qdevice(PKEY);
        qdevice(RKEY);
        qdevice(SKEY);
        qdevice(TKEY);
        qdevice(LEFTMOUSE);
        qdevice(RIGHTMOUSE);
        qdevice(MIDDLEMOUSE);
        qdevice(LEFTARROWKEY);
        qdevice(RIGHTARROWKEY);
        qdevice(UPARROWKEY);
```

```
  qdevice(DOWNARROWKEY);
  qdevice(HOMEKEY);
  qdevice(ENDKEY);
  qdevice(INSERTKEY);
  qdevice(PAGEUPKEY);
  qdevice(DELKEY);
  qdevice(PAGEDOWNKEY);
  qdevice(BUT52);  /* actually slash/question mark key */

  if (sim_control_.sim_type == GLmouse)
    {
      /* define the cursor (for mouse flying only) */
      curstype(C16X1);
      defcursor(1, cros);
      curorigin(1, 7, 7);
      setcursor(1, 0, 0);

      /* sleep(3);    /* to give us time to center cursor */

      /* set up to read mouse */
      getorigin(&org[X], &org[Y]);
      getsize(&size[X], &size[Y]);
      mdev[X] = MOUSEX;
      mdev[Y] = MOUSEY;

      mbias[X] = org[X]+size[X]/2;
      mbias[Y] = org[Y]+size[Y]/2;

      fscale[X] = LAT_SCALE/(double)(size[X]/2);
      fscale[Y] = LON_SCALE/(double)(size[Y]/2);
    }
  else
    {
      cursoff();
    }

  inited = TRUE;
}

do
  {
    abort = FALSE;
    if(qtest())
      {
        dev = qread(&val);
        if (val==0)
          {
            switch (dev)
              {
              case SPACEKEY:  /* trigger */
                trigger = -1;
                break;
              case ESCKEY:    /* abort */
                abort = TRUE;
                if (sim_control_.paused)
                  {
                    ls_resync(); /* turn timer back on */
                    sim_control_.paused = FALSE;
                  }
                break;
              case BUT52:  /* display help menu */
                ls_help();
                break;
              case AKEY:      /* retard throttle */
                Throttle_pct = Throttle_pct - 0.01;
                if(Throttle_pct <-0.2) Throttle_pct = -0.2;
                break;
              case SKEY:      /* advance throttle */
                Throttle_pct = Throttle_pct + 0.01;
                if(Throttle_pct >1.) Throttle_pct = 1;
                break;
              case LEFTMOUSE: /* left rudder */
                Rudder_pedal = Rudder_pedal + DELTARUDDER;
                break;
              case RIGHTMOUSE: /* right rudder */
                Rudder_pedal = Rudder_pedal - DELTARUDDER;
                break;
              case MIDDLEMOUSE: /* center rudder */
                Rudder_pedal = 0;
                break;
              case RKEY:      /* reset sim */
                ls_init();
                pushmatrix();
                drawworld(Phi*57.3, Theta*57.3, Psi*57.3,
                          X_pilot_rwy, Y_pilot_rwy, H_pilot_rwy, hudon );
                if(!sim_control_.debug) swapbuffers();
                popmatrix();
                break;
              case TKEY:      /* request trim */
                Q_body = 0.0; /* force to zero pitch rate */
                if (ls_trim()) ls_save_current_as_ic();
                break;
              case PKEY:      /* temporarily pause */
                if (sim_control_.paused)
                  {
                    ls_resync(); /* turn timer back on */
                    sim_control_.paused = FALSE;
                  }
                else
                  {
                    ls_unsync(); /* turn timer off to disable interrupts */
                    sim_control_.overrun = TRUE; /* will turn hud to red */
                    sim_control_.paused = TRUE;
                  }
                break;
              case LEFTARROWKEY:
                rotate(-DELTAARROWAMT, 'z');
                break;
              case RIGHTARROWKEY:
                rotate( DELTAARROWAMT, 'z');
                break;
              case UPARROWKEY:
                rotate( DELTAARROWAMT, 'y');
                break;
              case DOWNARROWKEY:
                rotate( -DELTAARROWAMT, 'y');
                break;
              case HOMEKEY:
                loadmatrix( mhome );
                break;
              case ENDKEY:
                loadmatrix( mhome );
                rotate( 1800, 'z');
                break;
              case INSERTKEY:
                loadmatrix( mhome );
                rotate( -900, 'z');
                break;
              case PAGEUPKEY:
                loadmatrix( mhome );
```

```
                rotate(  900, 'z');
                break;
              case DELKEY:
                loadmatrix( mhome );
                rotate( -450, 'y');
                rotate( -900, 'z');
                break;
              case PAGEDOWNKEY:
                loadmatrix( mhome );
                rotate( -450, 'y');
                rotate(  900, 'z');
                break;
              }
          }
        qreset();
      }

      if (sim_control_.sim_type == cockpit)
          {
            ls_ACES();    /* read stick, throttle, and buttons/switches */

            if( Left_button    > old_left_but    ) qenter(RKEY, 0);
            if( Right_button   > old_right_but   ) qenter(PKEY, 0);
            if( First_trigger  > old_first_trig  ) {};
            if( Second_trigger > old_second_trig ) {};
            old_left_but       = Left_button;
            old_right_but      = Right_button;
            old_first_trig     = First_trigger;
            old_second_trig    = Second_trigger;

            trigger = First_trigger;
          }
      else
          {
            getdev(2, mdev, mval );
            Long_control = fscale[Y]*(double)(mval[Y] - mbias[Y]);
            Lat_control  = fscale[X]*(double)(mval[X] - mbias[X]);
          }

      pushmatrix();
      drawworld(Phi*57.3, Theta*57.3, Psi*57.3,
              X_pilot_rwy, Y_pilot_rwy, H_pilot_rwy, hudon );
      gflush();
      popmatrix();
      if(!sim_control_.debug) swapbuffers();

    }
  while (sim_control_.paused);
  return abort;

}

void ls_cockpit_exit()
{
    gexit();
}
```

```
/****************************************************************************

        TITLE:          ls_ifterm.c

----------------------------------------------------------------------------

        FUNCTION:       UNIX curses terminal interface to LaRCsim

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 930202 by Bruce Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950226  Changed +a -s to -a +s to reflect correct sense of throttle
                keys; force simtype to terminal; added trim command 't';
                added header.                                   EBJ

        950329  Renamed from ls_ifsun.c v 1.4; cleaned up some. EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_ifterm.c,v 1.1 1995/03/29 16:04:19 bjax Exp $

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

static char rcsid[] = "$Id: ls_ifterm.c,v 1.1 1995/03/29 16:04:19 bjax Exp $";
```



```
#include <sys/time.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <curses.h>
#include <math.h>
#include <signal.h>
#include <stdio.h>
#include "ls_types.h"
#include "ls_generic.h"
#include "ls_constants.h"
#include "ls_sim_control.h"
#include "ls_cockpit.h"

#define TITLE 3
#define ESCKEY 0x1B

/* cockpit interface data block  - defn's in ls_cockpit.h */

#define aileron cockpit_.lat_stick
#define elevator cockpit_.long_stick
#define rudder cockpit_.rudder_pedal
#define throttle cockpit_.throttle_pct

COCKPIT cockpit_;

extern SCALAR Simtime; /* defined in ls_main */

static char *buf;

void drawhelp()
{
#define HELP 12
#define HELPCOL 20
  mvaddstr( HELP+0, HELPCOL, "              stick             ");
  mvaddstr( HELP+1, HELPCOL, "                                ");
  mvaddstr( HELP+2, HELPCOL, "                i               ");
  mvaddstr( HELP+3, HELPCOL, "throttle                   quit ");
  mvaddstr( HELP+4, HELPCOL, "                |               ");
  mvaddstr( HELP+5, HELPCOL, " -a  +s      j -k- l      <ESC> ");
  mvaddstr( HELP+6, HELPCOL, "                |               ");
  mvaddstr( HELP+7, HELPCOL, "                                ");
  mvaddstr( HELP+8, HELPCOL, "                <               ");
}

void drawpause()
{
  mvaddstr( HELP+0, HELPCOL, "                                ");
  mvaddstr( HELP+1, HELPCOL, "                                ");
  mvaddstr( HELP+2, HELPCOL, "                                ");
  mvaddstr( HELP+3, HELPCOL, "                                ");
  mvaddstr( HELP+4, HELPCOL, "     *** PAUSED (press P) ***   ");
  mvaddstr( HELP+5, HELPCOL, "                                ");
  mvaddstr( HELP+6, HELPCOL, "                                ");
  mvaddstr( HELP+7, HELPCOL, "                                ");
  mvaddstr( HELP+8, HELPCOL, "                                ");
}


int ls_cockpit()
{
        static int inited = 0;
        int nchr;
        int status;
        double sim_hr, sim_min, sim_sec;
```

```
if (inited==0)
{
        inited = -1;

        sim_control_.sim_type = terminal;

        buf = (char *) malloc(1000);
        initscr();
        cbreak();
        noecho();
        nonl();
        intrflush(stdscr,FALSE);
        keypad(stdscr,TRUE);

        move( TITLE, 3 );
        addstr( "L a R C S I M ");
        addstr(sim_control_.simname);

        mvaddstr( TITLE+2,  3, "Mach" );
        mvaddstr( TITLE+2, 18, "Psi" );
        mvaddstr( TITLE+2, 31, "NZ-G");
        mvaddstr( TITLE+3,  3, "KEAS");
        mvaddstr( TITLE+3, 18, "Thet");
        mvaddstr( TITLE+3, 31, "Alt" );
        mvaddstr( TITLE+4,  3, "Throt");
        mvaddstr( TITLE+4, 18, "Phi" );
        mvaddstr( TITLE+4, 31, "Hdot");
        mvaddstr( TITLE+3, 46, "Alpha");
        mvaddstr( TITLE+4, 46, "Beta");
        mvaddstr( TITLE+6,  3, "Elevator");
        mvaddstr( TITLE+6, 23, "Aileron" );
        mvaddstr( TITLE+6, 41, "Rudder");
        drawhelp();
        if (sim_control_.paused) drawpause();
}

move( TITLE,   50 );

sim_min = modf( Simtime/3600.0, &sim_hr )*60;
sim_sec = modf( sim_min, &sim_min )*60;

printw( "%01d:%02d:%04.1f", (int) sim_hr, (int) sim_min, sim_sec );
move( TITLE+4, 10 ); printw( "%5.0f %%", throttle*100);
move( TITLE+2,  9 ); printw( "%6.3f", Mach_number);
move( TITLE+3,  9 ); printw( "%6.1f", V_equiv_kts);
move( TITLE+2, 23 ); printw( "%5.1f", Psi*57.3);
move( TITLE+3, 23 ); printw( "%5.1f", Theta*57.3);
move( TITLE+4, 23 ); printw( "%5.1f", Phi*57.3);
move( TITLE+2, 36 ); printw( "%7.3f", -N_Z_pilot);
move( TITLE+3, 36 ); printw( "%7.0f", Altitude);
move( TITLE+4, 36 ); printw( "%7.3f", -V_down);
move( TITLE+3, 53 ); printw( "%5.2f", Alpha*57.3);
move( TITLE+4, 53 ); printw( "%5.2f", Beta*57.3);
move( TITLE+6, 15 ); printw( "%6.2f", elevator );
move( TITLE+6, 33 ); printw( "%6.2f", aileron );
move( TITLE+6, 51 ); printw( "%6.2f", rudder );

move( TITLE+10, 1 );
refresh();
status = ioctl(0, FIONBIO);  /* set I/O to non-blocking */
nchr = read(0, buf, 10);
while (nchr > 0)
    {
        nchr=0;
        switch (*buf)
            {
            case ESCKEY:
                return -1;
                break;
            case 'k':
                elevator = 0;
                aileron = 0;
                break;
            case 'i':
                elevator = elevator + 0.01;
                break;
            case ',':
                elevator = elevator - 0.01;
                break;
            case 'j':
                aileron =  aileron - 0.01;
                break;
            case 'l':
                aileron = aileron + 0.01;
                break;
            case 'a':
                throttle = throttle - 0.01;
                break;
            case 's':
                throttle = throttle + 0.01;
                break;
            case 'p':
                if (sim_control_.paused)
                    {
                        sim_control_.paused = FALSE;
                        drawhelp();
                        ls_resync(); /* turn timer back on */
                    }
                else
                    {
                        sim_control_.paused = TRUE;
                        drawpause();
                        sim_control_.overrun = TRUE; /* will turn hud to red
*/
                        ls_unsync(); /* turn timer off to disable interrupts
*/
                    }
                break;
            case 'r':
                ls_init();
                break;
            case 't':
                endwin();
                Q_body = 0.0;
                if (ls_trim()) ls_save_current_as_ic();
                break;
            }
    }
    return 0;
}


void ls_cockpit_exit()

{
  endwin();
  free(buf);
}
```

```
/***************************************************************************

        TITLE:  ls_init.c

----------------------------------------------------------------------------

        FUNCTION:       Initializes simulation

----------------------------------------------------------------------------

        MODULE STATUS:  incomplete

----------------------------------------------------------------------------

        GENEALOGY:      Written 921230 by Bruce Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    EBJ

        CODED BY:       EBJ

        MAINTAINED BY:  EBJ

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950314  Added get_set, put_set, and init routines.      EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_init.c,v 1.4 1995/03/15 12:15:23 bjax Stab $
$Log: ls_init.c,v $
 * Revision 1.4  1995/03/15  12:15:23  bjax
 * Added ls_init_get_set() and ls_init_put_set() and ls_init_init()
 * routines. EBJ
 *
 * Revision 1.3  1994/01/11  19:09:44  bjax
 * Fixed header includes.
 *
 * Revision 1.2  1992/12/30  14:04:53  bjax
 * Added call to ls_step(0, 1).
 *
 * Revision 1.1  92/12/30  14:02:19  bjax
 * Initial revision
 *
 * Revision 1.1  92/12/30  13:21:21  bjax
 * Initial revision
 *
 * Revision 1.3  93/12/31  10:34:11  bjax
 * Added $Log marker as well.
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
static char rcsid[] = "$Id: ls_init.c,v 1.4 1995/03/15 12:15:23 bjax Stab $";

#include <string.h>
#include <stdio.h>
#include "ls_types.h"
#include "ls_sym.h"

#define MAX_NUMBER_OF_CONTINUOUS_STATES 100
#define MAX_NUMBER_OF_DISCRETE_STATES  20
#define HARDWIRED 13
#define NIL_POINTER 0L

#define FACILITY_NAME_STRING "init"
#define CURRENT_VERSION 10

typedef struct
{
    symbol_rec  Symbol;
    double      value;
} cont_state_rec;

typedef struct
{
    symbol_rec  Symbol;
    long        value;
} disc_state_rec;


extern SCALAR Simtime;

static int Symbols_loaded = 0;
static int Number_of_Continuous_States = 0;
static int Number_of_Discrete_States = 0;
static cont_state_rec   Continuous_States[ MAX_NUMBER_OF_CONTINUOUS_STATES ];
static disc_state_rec   Discrete_States[  MAX_NUMBER_OF_DISCRETE_STATES ];


void ls_init_init()
{
    int i, error;

    if (Number_of_Continuous_States == 0)
        {
            Number_of_Continuous_States = HARDWIRED;

            for (i=0;i<HARDWIRED;i++)
                strcpy( Continuous_States[i].Symbol.Mod_Name, "**" );

            strcpy( Continuous_States[ 0].Symbol.Par_Name,
                    "generic_.geodetic_position_v[0]");
            strcpy( Continuous_States[ 1].Symbol.Par_Name,
                    "generic_.geodetic_position_v[1]");
            strcpy( Continuous_States[ 2].Symbol.Par_Name,
```

```
                    "generic_.geodetic_position_v[2]");
            strcpy( Continuous_States[ 3].Symbol.Par_Name,
                    "generic_.v_local_v[0]");
            strcpy( Continuous_States[ 4].Symbol.Par_Name,
                    "generic_.v_local_v[1]");
            strcpy( Continuous_States[ 5].Symbol.Par_Name,
                    "generic_.v_local_v[2]");
            strcpy( Continuous_States[ 6].Symbol.Par_Name,
                    "generic_.euler_angles_v[0]");
            strcpy( Continuous_States[ 7].Symbol.Par_Name,
                    "generic_.euler_angles_v[1]");
            strcpy( Continuous_States[ 8].Symbol.Par_Name,
                    "generic_.euler_angles_v[2]");
            strcpy( Continuous_States[ 9].Symbol.Par_Name,
                    "generic_.omega_body_v[0]");
            strcpy( Continuous_States[10].Symbol.Par_Name,
                    "generic_.omega_body_v[1]");
            strcpy( Continuous_States[11].Symbol.Par_Name,
                    "generic_.omega_body_v[2]");
            strcpy( Continuous_States[12].Symbol.Par_Name,
                    "generic_.earth_position_angle");
        }

    for (i=0;i<Number_of_Continuous_States;i++)
        {
            (void) ls_get_sym_val( &Continuous_States[i].Symbol, &error );
            if (error) Continuous_States[i].Symbol.Addr = NIL_POINTER;
        }

    for (i=0;i<Number_of_Discrete_States;i++)
        {
            (void) ls_get_sym_val( &Discrete_States[i].Symbol, &error );
            if (error) Discrete_States[i].Symbol.Addr = NIL_POINTER;
        }
}

void ls_init()
{
    int i;

    Simtime = 0;

    ls_init_init();
    /* move the states to proper values */

    for(i=0;i<Number_of_Continuous_States;i++)
        if (Continuous_States[i].Symbol.Addr)
            ls_set_sym_val( &Continuous_States[i].Symbol,
                            Continuous_States[i].value );

    for(i=0;i<Number_of_Discrete_States;i++)
        if (Discrete_States[i].Symbol.Addr)
            ls_set_sym_val( &Discrete_States[i].Symbol,
                            (double) Discrete_States[i].value );


    model_init();

    ls_step(0.0, -1);
}

char *ls_init_get_set(char *buffer, char *eob)
/* This routine parses the settings file for "init" entries. */
{

    static char *fac_name = FACILITY_NAME_STRING;
    char *bufptr, **lasts, *nullptr, null = '\0';
    char line[256];
    int n, ver, i, error, abrt;

    enum {cont_states_header, cont_states, disc_states_header, disc_states, done } l
ooking_for;

    nullptr = &null;
    lasts = &nullptr;
    abrt = 0;
    looking_for = cont_states_header;

    n = sscanf(buffer, "%s", line);
    if (n == 0) return 0L;
    if (strncasecmp( fac_name, line, strlen(fac_name) )) return 0L;

    bufptr = strtok_r( buffer+strlen(fac_name)+1, "\n", lasts);
    if (bufptr == 0) return 0L;

    sscanf( bufptr, "%d", &ver );
    if (ver != CURRENT_VERSION) return 0L;

    ls_init_init();

    while( !abrt && (eob > bufptr))
      {
        bufptr = strtok_r( 0L, "\n", lasts );
        if (bufptr == 0) return 0L;
        if (strncasecmp( bufptr, "end", 3) == 0) break;

        sscanf( bufptr, "%s", line );
        if (line[0] != '#') /* ignore comments */
          {
            switch (looking_for)
              {
                case cont_states_header:
                  {
                    if (strncasecmp( line, "continuous_states", 17) == 0)
                      {
                        n = sscanf( bufptr, "%s%d", line,
                                    &Number_of_Continuous_States );
                        if (n != 2) abrt = 1;
                        looking_for = cont_states;
                        i = 0;
                      }
                    break;
                  }
                case cont_states:
                  {
                    n = sscanf( bufptr, "%s%s%le",
                                Continuous_States[i].Symbol.Mod_Name,
                                Continuous_States[i].Symbol.Par_Name,
                                &Continuous_States[i].value );
                    if (n != 3) abrt = 1;
                    Continuous_States[i].Symbol.Addr = NIL_POINTER;
                    i++;
                    if (i >= Number_of_Continuous_States)
                        looking_for = disc_states_header;
                    break;
                  }
                case disc_states_header:
                  {
                    if (strncasecmp( line, "discrete_states", 15) == 0)
```


C-2

```
                {
                    n = sscanf( bufptr, "%s%d", line,
                                &Number_of_Discrete_States );
                    if (n != 2) abrt = 1;
                    looking_for = disc_states;
                    i = 0;
                }
                break;
            }
        case disc_states:
            {
                n = sscanf( bufptr, "%s%s%ld",
                            Discrete_States[i].Symbol.Mod_Name,
                            Discrete_States[i].Symbol.Par_Name,
                            &Discrete_States[i].value );
                if (n != 3) abrt = 1;
                Discrete_States[i].Symbol.Addr = NIL_POINTER;
                i++;
                if (i >= Number_of_Discrete_States) looking_for = done;
            }
        case done:
            {
                break;
            }
        }

      }
   }

   Symbols_loaded = !abrt;

   bufptr = *lasts;
   return bufptr;
}


void ls_init_put_set( FILE *fp )
{
    int i;

    if (fp==0) return;
    fprintf(fp, "\n");
    fprintf(fp, "#==============================  %s\n", FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, "%04d\n", CURRENT_VERSION);
    fprintf(fp, "  continuous_states: %d\n", Number_of_Continuous_States);
    fprintf(fp, "#    module    parameter   value\n");
    for (i=0; i<Number_of_Continuous_States; i++)
        fprintf(fp, "    %s\t%s\t%E\n",
                Continuous_States[i].Symbol.Mod_Name,
                Continuous_States[i].Symbol.Par_Name,
                Continuous_States[i].value );

    fprintf(fp, "  discrete_states: %d\n", Number_of_Discrete_States);
    fprintf(fp, "#    module    parameter   value\n");
    for (i=0;i<Number_of_Discrete_States;i++)
        fprintf(fp, "    %s\t%s\t%ld\n",
                Discrete_States[i].Symbol.Mod_Name,
                Discrete_States[i].Symbol.Par_Name,
                Discrete_States[i].value );
    fprintf(fp, "end\n");
    return;
}

void ls_save_current_as_ic()
{
    /* Save current states as initial conditions */

    int i, error;

    for(i=0;i<Number_of_Continuous_States;i++)
        if (Continuous_States[i].Symbol.Addr)
            Continuous_States[i].value =
                ls_get_sym_val( &Continuous_States[i].Symbol, &error );

    for(i=0;i<Number_of_Discrete_States;i++)
        if (Discrete_States[i].Symbol.Addr)
            Discrete_States[i].value = (long)
                ls_get_sym_val( &Discrete_States[i].Symbol, &error );

}
```

```
/***************************************************************************

        TITLE:          ls_matrix.c

----------------------------------------------------------------------------

        FUNCTION:       general real matrix routines; includes
                                gaussj() for matrix inversion using
                                Gauss-Jordan method with full pivoting.

        The routines in this module have come more or less from ref [1].
        Note that, probably due to the heritage of ref [1] (which has a
        FORTRAN version that was probably written first), the use of 1 as
        the first element of an array (or vector) is used. This is accomplished
        in memory by allocating, but not using, the 0 elements in each dimension.
        While this wastes some memory, it allows the routines to be ported more
        easily from FORTRAN (I suspect) as well as adhering to conventional
        matrix notation.  As a result, however, traditional ANSI C convention
        (0-base indexing) is not followed; as the authors of ref [1] point out,
        there is some question of the portability of the resulting routines
        which sometimes access negative indexes. See ref [1] for more details.

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 950222 E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    from Numerical Recipes in C, by Press, et. al.

        CODED BY:       Bruce Jackson

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_matrix.c,v 1.1 1995/02/27 19:55:44 bjax Stab $
$Log: ls_matrix.c,v $
 * Revision 1.1  1995/02/27  19:55:44  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:     [1] Press, William H., et. al, Numerical Recipes in
                            C, 2nd edition, Cambridge University Press, 1992

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "ls_matrix.h"

#define SWAP(a,b) (temp=(a);(a)=(b);(b)=temp;)

static char rcsid[] = "$Id: ls_matrix.c,v 1.1 1995/02/27 19:55:44 bjax Stab $";


int *nr_ivector(long nl, long nh)
{
    int *v;

    v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
    return v-nl+NR_END;
}



double **nr_matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
    long i, nrow=nrh-nrl+1, ncol=nch-ncl+1;
    double **m;

    /* allocate pointers to rows */
    m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));

    if (!m)
        {
            fprintf(stderr, "Memory failure in routine 'nr_matrix'.\n");
            exit(1);
        }

    m += NR_END;
    m -= nrl;

    /* allocate rows and set pointers to them */
    m[nrl] = (double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
    if (!m[nrl])
        {
            fprintf(stderr, "Memory failure in routine 'matrix'\n");
            exit(1);
        }

    m[nrl] += NR_END;
    m[nrl] -= ncl;

    for (i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

    /* return pointer to array of pointers to rows */
    return m;
}
```

```c
void nr_free_ivector(int *v, long nl, long nh)
{
    free( (char *) (v+nl-NR_END));
}


void nr_free_matrix(double **m, long nrl, long nrh, long ncl, long nch)
/* free a double matrix allocated by nr_matrix() */
{
    free((char *) (m[nrl]+ncl-NR_END));
    free((char *) (m+nrl-NR_END));
}


int nr_gaussj(double **a, int n, double **b, int m)

/* Linear equation solution by Gauss-Jordan elimination. a[1..n][1..n] is */
/* the input matrix. b[1..n][1..m] is input containing the m right-hand   */
/* side vectors. On output, a is replaced by its matrix invers, and b is  */
/* replaced by the corresponding set of solution vectors.                 */

/* Note: this routine modified by EBJ to make b optional, if m == 0 */

{
    int         *indxc, *indxr, *ipiv;
    int         i, icol, irow, j, k, l, ll;
    double      big, dum, pivinv, temp;

    int         bexists = ((m != 0) || (b == 0));

    indxc = nr_ivector(1,n);    /* The integer arrays ipiv, indxr, and  */
    indxr = nr_ivector(1,n);    /* indxc are used for pivot bookkeeping */
    ipiv  = nr_ivector(1,n);

    for (j=1;j<=n;j++) ipiv[j] = 0;

    for (i=1;i<=n;i++)          /* This is the main loop over columns   */
        {
            big = 0.0;
            for (j=1;j<=n;j++)  /* This is outer loop of pivot search   */
                if (ipiv[j] != 1)
                    for (k=1;k<=n;k++)
                        {
                            if (ipiv[k] == 0)
                                {
                                    if (fabs(a[j][k]) >= big)
                                        {
                                            big = fabs(a[j][k]);
                                            irow = j;
                                            icol = k;
                                        }
                                }
                            else
                                if (ipiv[k] > 1) return -1;
                        }
            ++(ipiv[icol]);

/*    We now have the pivot element, so we interchange rows, if needed,  */
/* to put the pivot element on the diagonal.  The columns are not        */
/* physically interchanged, only relabeled: indxc[i], the column of the  */
/* ith pivot element, is the ith column that is reduced, while indxr[i]  */
/* is the row in which that pivot element was orignally located. If      */
/* indxr[i] != indxc[i] there is an implied column interchange.  With    */
/* this form of bookkeeping, the solution b's will end up in the correct */
/* order, and the inverse matrix will be scrambed by columns.            */

            if (irow != icol)
                {
/*                  for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l]) */
                    for (l=1;l<=n;l++)
                      {
                            temp=a[irow][l];
                            a[irow][l]=a[icol][l];
                            a[icol][l]=temp;
                      }
                    if (bexists) for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
                }
            indxr[i] = irow;    /* We are now ready to divide the pivot row */
            indxc[i] = icol;    /* by the pivot element, a[irow][icol]      */
            if (a[icol][icol] == 0.0) return -1;
            pivinv = 1.0/a[icol][icol];
            a[icol][icol] = 1.0;
            for (l=1;l<=n;l++) a[icol][l] *= pivinv;
            if (bexists) for (l=1;l<=m;l++) b[icol][l] *= pivinv;
            for (ll=1;ll<=n;ll++)       /* Next, we reduce the rows...  */
                if (ll != icol )        /* .. except for the pivot one  */
                    {
                        dum = a[ll][icol];
                        a[ll][icol] = 0.0;
                        for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
          if (bexists) for (l=1;l<=m;l++) b[ll][i] -= b[icol][l]*dum;
                    }
        }

/* This is the end of the mail loop over columns of the reduction. It
       only remains to unscrambled the solution in view of the column
       interchanges. We do this by interchanging pairs of columns in
       the reverse order that the permutation was built up. */

    for (l=n;l>=1;l--)
        {
            if (indxr[l] != indxc[l])
                for (k=1;k<=n;k++)
                    SWAP(a[k][indxr[l]],a[k][indxc[l]])
        }

/* and we are done */

    nr_free_ivector(ipiv,1,n);
    nr_free_ivector(indxr,1,n);
    nr_free_ivector(indxc,1,n);

    return 0;   /* indicate success */
}

void nr_copymat(double **orig, int n, double **copy)
/* overwrites matrix 'copy' with copy of matrix 'orig' */
{
        long i, j;

        if ((orig==0)||(copy==0)||(n==0)) return;

        for (i=1;i<=n;i++)
                for (j=1;j<=n;j++)
                        copy[i][j] = orig[i][j];
}

void nr_multmat(double **m1, int n, double **m2, double **prod)
{
        long i, j, k;
```

```
        if ((m1==0)||(m2==0)||(prod==0)||(n==0)) return;

        for (i=1;i<=n;i++)
                for (j=1;j<=n;j++)
                        {
                                prod[i][j] = 0.0;
                                for(k=1;k<=n;k++) prod[i][j] += m1[i][k]*m2[k][j];
                        }
}


void nr_printmat(double **a, int n)
{
    int i,j;

    printf("\n");
    for(i=1;i<=n;i++)
      {
        for(j=1;j<=n;j++)
            printf("% 9.4f ", a[i][j]);
        printf("\n");
      }
    printf("\n");

}


testmat() /* main() /* for test purposes */
{
    double **mat1, **mat2, **mat3;
    double invmaxlong;
    int loop, i, j, n = 20;
    long maxlong = RAND_MAX;
    int maxloop = 2;

    invmaxlong = 1.0/(double)maxlong;
    mat1 = nr_matrix(1, n, 1, n );
    mat2 = nr_matrix(1, n, 1, n );
    mat3 = nr_matrix(1, n, 1, n );

/*    for(i=1;i<=n;i++) mat1[i][i]= 5.0; */

        for(loop=0;loop<maxloop;loop++)
        {
            if (loop != 0)
                for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++)
                        mat1[i][j] = 2.0 - 4.0*invmaxlong*(double) rand();

                printf("Original matrix:\n");
            nr_printmat( mat1, n );

            nr_copymat( mat1, n, mat2 );

            i = nr_gaussj( mat2, n, 0, 0 );

            if (i) printf("Singular matrix.\n");

                printf("Inverted matrix:\n");
            nr_printmat( mat2, n );

            nr_multmat( mat1, n, mat2, mat3 );

                printf("Original multiplied by inverse:\n");
            nr_printmat( mat3, n );

            if (loop < maxloop-1) sleep(1);
        }

    nr_free_matrix( mat1, 1, n, 1, n );
    nr_free_matrix( mat2, 1, n, 1, n );
    nr_free_matrix( mat3, 1, n, 1, n );

}
```

```
/***************************************************************************

        TITLE:          ls_model()

----------------------------------------------------------------------------

        FUNCTION:       Model loop executive

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 15 October 1992 as part of LaRCSIM project
                        by Bruce Jackson.

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:  maintainer

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950306  Added parameters to call: dt, which is the step size
                to be taken this loop (caution: may vary from call to call)
                and Initialize, which if non-zero, implies an initialization
                is requested.                                   EBJ

        CURRENT RCS HEADER INFO:
$Header: /aces/larcsim/dev/RCS/ls_model.c,v 1.3 1995/03/06 18:49:46 bjax Stab $
$Log: ls_model.c,v $
 * Revision 1.3  1995/03/06  18:49:46  bjax
 * Added dt and initialize flag parameters to subroutine calls. This will
 * support trim routine (to allow single throttle setting to drive
 * all four throttle positions, for example, if initialize is TRUE).
 *
 * Revision 1.2  1993/03/10  06:38:09  bjax
 * Added additional calls: inertias() and subsystems()... EBJ
 *
 * Revision 1.1  92/12/30  13:19:08  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:      ls_step (in initialization), ls_loop (planned)

----------------------------------------------------------------------------

        CALLS TO:       aero(), engine(), gear()

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
#include "ls_types.h"

void ls_model( SCALAR dt, int Initialize )

{
  inertias( dt, Initialize );
  subsystems( dt, Initialize );
  aero( dt, Initialize );
  engine( dt, Initialize );
  gear( dt, Initialize );
}
```

```
/*****************************************************************************

	TITLE:		ls_record	$Id: ls_record.c,v 1.11 1995/04/07 01:46:43 bjax Exp
 $

------------------------------------------------------------------------------

	FUNCTION:	Store time history data from sim runs

------------------------------------------------------------------------------

	MODULE STATUS:	developmental

------------------------------------------------------------------------------

	GENEALOGY:	Created October 26, 1992 by Bruce Jackson

------------------------------------------------------------------------------

	DESIGNED BY:	Bruce Jackson

	CODED BY:	Bruce Jackson

	MAINTAINED BY:	maintainer

------------------------------------------------------------------------------

	MODIFICATION HISTORY:

	DATE	PURPOSE						BY

	930727	Original version (1.1) was hardwired to record certain variables;
		this version uses the ls_sym routines to look up scalar addresses
		prior to run. (Note: this doesn't work right yet for many of the
		variables defined in ls_eom.h, since they are only definitions
		of three-element vectors... shucks).
							EBJ

	931008	Added "interp"olation of frames, so only every INTERP'd frame
		is retained. Note that, by design, data is recorded every frame
		so that execution times don't vary; but only every INTERP'th frame
		is saved (by advancing the Next pointer).  This automatically saves
		the very first frame and the very last frame,  although the last
		frame won't necessarily be spaced equally with regard to preceeding
		frames.						EBJ

	931008	Also added min and max value info for each channel. EBJ

	940111	Changed include files from ls_eom.h to ls_types,  ls_generic,  and
		ls_sim_control.h

	940121	Modified to use new, improved ls_findsym, which now understands and
		locates structures and arrays (both of which are needed to access
		the new global variables).				EBJ

	940506	Corrected logic that was setting value of each parm to zero. EBJ
		Also gave fixed variable useful names.

	940509	Fixed bug in circular buffer handling; now buffer wraps to
		location 0 correctly.

	950228	Incorporated utility routines provided now by ls_sym, such as
		ls_print_findsym_error() and ls_get_double().			EBJ

	950302	Modified CHANNEL definition in ls_tape.h to use symbol_rec provided
		by ls_sym.h to get consistent use of symbol record in this facility.

	950307	Now supports ls_record_get_set() and ls_record_put_set() calls. EBJ

	950405	Made length of data channels a sim_control_ parameter.		EBJ

	CURRENT RCS HEADER INFO:

$Header: /aces/larcsim/dev/RCS/ls_record.c,v 1.11 1995/04/07 01:46:43 bjax Exp $
$Log: ls_record.c,v $
 * Revision 1.11  1995/04/07  01:46:43  bjax
 * Modified to use Tape->Length instead of MAX_SLICES as tape length
 * reference; changed ls_record_tape_init() into two parts: ls_record_
 * channels_init() and ls_record_alloc_storage(), so individual Channels
 * can be allocated at the last moment, when Tape->Length has been determined;
 * modified initialization of Tape structure to reflect change of Channel
 * .Data from array of SCALAR to point to array of scalars.
 *
 * Revision 1.10  1995/03/15  12:16:23  bjax
 * Added flag marker line to ls_record_put_set() routine.
 *
 * Revision 1.9  1995/03/07  22:36:06  bjax
 * Moved short names of hardwired variables to alias field; added ls_record_put_set(
)
 * function. EBJ
 *
 * Revision 1.8  1995/03/06  18:42:34  bjax
 * Major structural changes: making use of ls_get_sym_val; separated
 * ls_record_tape_init() from ls_record() body; added ls_record_get_set();
 * minor cleanups.
 *
 * Revision 1.7  1995/03/03  02:00:50  bjax
 * Modified to use new def'n of Tape->Chan structure (includes symbol rec
 * defined in ls_sym.h). EBJ
 *
 * Revision 1.6  1995/02/28  12:58:16  bjax
 * Modified to use new ls_sym routines ls_print_findsym_error
 * and ls_get_double().  EBJ
 *
 * Revision 1.5  1994/05/17  12:25:05  bjax
 * For unknown reasons, the "interp" initialization was being
 * incorrectly initialized by sim_control_.save_settings. Changed
 * the declaration from static int to static short fixed the problem,
 * it appears.
 *
 * Revision 1.4  1994/05/09  21:20:52  bjax
 * Fixed problem with tape wrapping to second time slice.
 *
 * Revision 1.3  1994/05/06  20:18:27  bjax
 * More or less complete set of data types now converted properly.
 * Added comment line (first column '#') in .set settings file.
 *
 * Revision 1.2.1.13  1994/05/06  18:22:46  bjax
 * Gave useful short names to fixed data channels 0-18; corrected
 * interpretation and conversion of most data types.
 *
 * Revision 1.2.1.12  1994/05/06  16:35:48  bjax
 * Added close of settings file to end of ls_record_get_set routine.
 *
 * Revision 1.2.1.11  1994/05/06  16:32:02  bjax
 * Minor mods to record_get_set routine.
 *
 * Revision 1.2.1.10  1994/05/06  15:32:24  bjax
 * Fixed bug with all data values set to zero.
 *
```

```
 * Revision 1.2.1.9  1994/03/28  19:43:38  bjax
 * Added support for local "settings" file (e.g. navion.set) in cwd.
 * There appears to be a problem in ls_findsym, however.
 *
 * Revision 1.2.1.8  1994/01/11  18:56:41  bjax
 * Changed header includes from ls_eom to ls_types, ls_generic, and ls_sim_control
 *
 * Revision 1.2.1.7  1993/12/20  16:50:48  bjax
 * Cleaned up the time slice acess method. EBJ
 *
 * Revision 1.2.1.6  1993/10/08  22:04:02  bjax
 * Added Min value, max value calculations at record time. EBJ
 *
 * Revision 1.2.1.5  1993/10/08  19:34:36  bjax
 * Added interpolation logic, so every frame 6th frame is saved (10 Hz
 * at present execution speeds). Need to make it a control variable,
 * tough.  -- EBJ
 *
 * Revision 1.2.1.4  1993/08/03  20:00:09  bjax
 * Fixed to make Tape.Chan[].Addr pointer type compatible with ls_findsym call.
 *
 * Revision 1.2.1.3  1993/07/30  18:33:57  bjax
 * Corrected index on rudder initialization.
 *
 * Revision 1.2.1.1  1993/07/28  16:22:26  bjax
 * Further development of using symbol table lookups. EBJ
 *
 * Revision 1.1  1992/12/30  13:19:27  bjax
 * Initial revision
 *

--------------------------------------------------------------------------

        REFERENCES:

--------------------------------------------------------------------------

        CALLED BY:

--------------------------------------------------------------------------

        CALLS TO:

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/


#include "ls_types.h"
#include "ls_generic.h"
#include "ls_sim_control.h"
#include "ls_tape.h" /* includes ls_sym.h */
#include <math.h>
#include <string.h> /* for strtok, strncmp, etc. */
#include <stdio.h>

#define NIL_POINTER 0L
#define INTERP sim_control_.save_spacing
#define LINE_LENGTH 256

#define HARDWIRED 19

#define FACILITY_NAME_STRING "record"
#define CURRENT_VERSION 10

extern TAPE *Tape;      /* declared in ls_main.c */
extern SCALAR Simtime;  /* declared in ls_main.c */

static int noTape = 0;
char line[LINE_LENGTH];
int num, abort;
SYMBOL_NAME mod_name, par_name;


ls_record_channels_init()
{
    int i, result;
    static int channels_allocated = 0;

    if (!channels_allocated)
      {
        channels_allocated = -1;
        noTape = 0;

        Tape = (TAPE *) malloc(sizeof(TAPE));
        if (Tape == NIL_POINTER)
        {
            fprintf(stderr, "ls_tape: memory allocation error\n");
            noTape = -1;
            return;
        } /* end of "if (Tape == NIL_POINTER)" path when
             allocating Tape structure */
        else
        {
            Tape->Num_Chan = HARDWIRED;  /* presets */
            Tape->First = -1;
            Tape->Last = 0;
            Tape->Current = 0;
            Tape->Next = 0;
            for(i=0;i<Tape->Num_Chan;i++)
            {
                Tape->Chan[i] = (CHANNEL *) malloc( sizeof( CHANNEL ));
                if (Tape->Chan[i] == NIL_POINTER)
                {
                    fprintf(stderr, "ls_tape: memory allocation error\n");
                    noTape = -1;
                    for(;i>0;--i) free(Tape->Chan[i]);
                    free(Tape);
                    return;
                } /* end of "if (Tape->Chan[i] == NIL_POINTER)" path */
                else
                {
                    Tape->Chan[i]->Max_value = (SCALAR) sqrt(-3.); /* NaN */
                    Tape->Chan[i]->Min_value = (SCALAR) sqrt(-3.);
                } /* end of "if (Tape->Chan[i] <> NIL_POINTER)" path */
            } /* end of "for(i=0;i<Tape->Num_Chan;++)" loop */
            for(i=0;i<16;i++)
              {
                strcpy( Tape->Chan[i]->Symbol.Mod_Name, "*" );
                      /* first few are global */
              }
            strcpy( Tape->Chan[ 0]->Symbol.Par_Name,
                    "generic_.geocentric_position_v[0]");
            strcpy( Tape->Chan[ 1]->Symbol.Par_Name,
                    "generic_.geocentric_position_v[1]");
```

```
        strcpy( Tape->Chan[ 2]->Symbol.Par_Name,
                "generic_.geocentric_position_v[2]");
        strcpy( Tape->Chan[ 3]->Symbol.Par_Name,
                "generic_.v_local_v[0]");
        strcpy( Tape->Chan[ 4]->Symbol.Par_Name,
                "generic_.v_local_v[1]");
        strcpy( Tape->Chan[ 5]->Symbol.Par_Name,
                "generic_.v_local_v[2]");
        strcpy( Tape->Chan[ 6]->Symbol.Par_Name,
                "generic_.euler_angles_v[0]");
        strcpy( Tape->Chan[ 7]->Symbol.Par_Name,
                "generic_.euler_angles_v[1]");
        strcpy( Tape->Chan[ 8]->Symbol.Par_Name,
                "generic_.euler_angles_v[2]");
        strcpy( Tape->Chan[ 9]->Symbol.Par_Name,
                "generic_.omega_body_v[0]");
        strcpy( Tape->Chan[10]->Symbol.Par_Name,
                "generic_.omega_body_v[1]");
        strcpy( Tape->Chan[11]->Symbol.Par_Name,
                "generic_.omega_body_v[2]");
        strcpy( Tape->Chan[12]->Symbol.Par_Name,
                "generic_.earth_position_angle");
        strcpy( Tape->Chan[13]->Symbol.Par_Name,
                "generic_.d_cg_rwy_local_v[0]");
        strcpy( Tape->Chan[14]->Symbol.Par_Name,
                "generic_.d_cg_rwy_local_v[1]");
        strcpy( Tape->Chan[15]->Symbol.Par_Name,
                "generic_.d_cg_rwy_local_v[2]");

    /* control positions hardwired as well */

        strcpy( Tape->Chan[16]->Symbol.Mod_Name, "*");
        strcpy( Tape->Chan[16]->Symbol.Par_Name, "cockpit_.long_stick");
        strcpy( Tape->Chan[17]->Symbol.Mod_Name, "*");
        strcpy( Tape->Chan[17]->Symbol.Par_Name, "cockpit_.lat_stick");
        strcpy( Tape->Chan[18]->Symbol.Mod_Name, "*");
        strcpy( Tape->Chan[18]->Symbol.Par_Name, "cockpit_.rudder_pedal");

        for(i=0;i<Tape->Num_Chan;i++)
        {
            result = ls_findsym( Tape->Chan[i]->Symbol.Mod_Name,
                                 Tape->Chan[i]->Symbol.Par_Name,
                      (char **) &Tape->Chan[i]->Symbol.Addr,
                                &Tape->Chan[i]->Symbol.Par_Type  );
            if (result)
              {
                  ls_print_findsym_error( result,
                                          Tape->Chan[i]->Symbol.Mod_Name,
                                          Tape->Chan[i]->Symbol.Par_Name );
                  Tape->Chan[i]->Symbol.Addr = NIL_POINTER;
              } /* end of result non-zero path */
          } /* end of "for(i=0;i<Tape->Num_Chan;i++)" loop */

          /* now rename fixed channels to something useful */
          strcpy( Tape->Chan[ 0]->Symbol.Alias,  "geocLat");
          strcpy( Tape->Chan[ 1]->Symbol.Alias,  "geocLon");
          strcpy( Tape->Chan[ 2]->Symbol.Alias,  "radiusV");
          strcpy( Tape->Chan[ 3]->Symbol.Alias,  "v_north");
          strcpy( Tape->Chan[ 4]->Symbol.Alias,  "v_east");
          strcpy( Tape->Chan[ 5]->Symbol.Alias,  "v_down");
          strcpy( Tape->Chan[ 6]->Symbol.Alias,  "phi_r");
          strcpy( Tape->Chan[ 7]->Symbol.Alias,  "theta_r");
          strcpy( Tape->Chan[ 8]->Symbol.Alias,  "psi_r");
          strcpy( Tape->Chan[ 9]->Symbol.Alias,  "pb");
          strcpy( Tape->Chan[10]->Symbol.Alias,  "qb");
          strcpy( Tape->Chan[11]->Symbol.Alias,  "rb");
          strcpy( Tape->Chan[12]->Symbol.Alias,  "epa");
          strcpy( Tape->Chan[13]->Symbol.Alias,  "X");
          strcpy( Tape->Chan[14]->Symbol.Alias,  "Y");
          strcpy( Tape->Chan[15]->Symbol.Alias,  "Z");
          strcpy( Tape->Chan[16]->Symbol.Alias,  "long_stk");
          strcpy( Tape->Chan[17]->Symbol.Alias,  "lat_stk");
          strcpy( Tape->Chan[18]->Symbol.Alias,  "rud_ped");
        } /* end of "(Tape != NIL_POINTER)" path */
      } /* end of "(!channels_allocated)" path */

} /* end of ls_record_channels_init() */


void ls_record_alloc_storage()
{
    int i;
    static int storage_allocated = 0;

    if (!storage_allocated && !noTape)
        {
            Tape->Length = sim_control_.time_slices;
            Tape->T_Stamp = (SCALAR *) malloc( Tape->Length*sizeof( SCALAR ) );
            if (Tape->T_Stamp == NIL_POINTER)
                {
                    fprintf(stderr, "ls_tape: memory allocation error\n");
                    for(i=0;i<Tape->Num_Chan;i++) free(Tape->Chan[i]);
                    free(Tape);
                    noTape = -1;
                    return;
                } /* end of "if (Tape->T_Stamp == NIL_POINTER)" path */

            for(i=0;i<Tape->Num_Chan;i++)
                {
                    Tape->Chan[i]->Data = (SCALAR *) malloc( Tape->Length*sizeof( SC
ALAR ) );
                    if (Tape->Chan[i]->Data == NIL_POINTER)
                        {
                            fprintf(stderr, "ls_tape: memory allocation error\n");
                            noTape = -1;
                            free(Tape->Chan[i]);
                            for(;i>0;--i) {free(Tape->Chan[i]->Data); free(Tape->Cha
n[i]);}
                            free(Tape);
                            return;
                        } /* end of "if (Tape->Chan[i]->Data == NIL_POINTER)" path *
/
                } /* end of for(i=0;i<Tape->Num_Chan;i++) loop */
        }
}

char *ls_record_get_set(char *buffer, char *eob)
/* This routine parses the settings file for "tape" entries. */
{

    static char *fac_name = FACILITY_NAME_STRING;
    char *bufptr, **lasts, *nullptr, null = '\0';
    int n, ver, i, error;

    nullptr = &null;
    lasts = &nullptr;
    abort = 0;


    n = sscanf(buffer, "%s", line);
```

```
    if (n == 0) return 0L;
    if (strncasecmp( fac_name, line, strlen(fac_name) )) return 0L;

    bufptr = strtok_r( buffer+strlen(fac_name)+1, "\n", lasts);
    if (bufptr == 0) return 0L;

    sscanf( bufptr, "%d", &ver );
    if (ver != CURRENT_VERSION) return 0L;

    ls_record_channels_init();

    while( !abort && (eob > bufptr))
      {
        bufptr = strtok_r( 0L, "\n", lasts );
        if (bufptr == 0) return 0L;
        if (strncasecmp( bufptr, "end", 3) == 0) break;

        sscanf( bufptr, "%s", line );
        if (line[0] != '#') /* ignore comments */
          {
            num = sscanf( bufptr, "%s %s", mod_name, par_name );
            if ((num == 2) && (Tape->Num_Chan < MAX_TAPE_CHANNELS))
                {       /* add channel */
                    i = Tape->Num_Chan;
                    Tape->Chan[i] = (CHANNEL *) malloc( sizeof( CHANNEL ) );
                    if (Tape->Chan[Tape->Num_Chan] == NIL_POINTER)
                        {
                            fprintf(stderr, "ls_tape: memory allocation error\n");
                            abort = -1;
                        }
                    else
                        {

                            /* initialize to NaN */
                            Tape->Chan[i]->Max_value = (SCALAR) sqrt(-3.);
                            Tape->Chan[i]->Min_value = (SCALAR) sqrt(-3.);
                            strcpy( Tape->Chan[i]->Symbol.Mod_Name,
                                    mod_name);
                            strcpy( Tape->Chan[i]->Symbol.Par_Name,
                                    par_name);
                            (void) ls_get_sym_val( &Tape->Chan[i]->Symbol, &error );
                            if (error) Tape->Chan[i]->Symbol.Addr = 0L;
                            Tape->Num_Chan++;

                        }
                }
          }
      }
    bufptr = *lasts;
    return bufptr;
}


void ls_record_put_set( FILE *fp )
{
    int i;

    if (fp==0) return;
    fprintf(fp, "\n");
    fprintf(fp, "#==============================  %s\n", FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, "%04d\n", CURRENT_VERSION);
    for (i=HARDWIRED; i<Tape->Num_Chan; i++)
        fprintf(fp, "    %s     %s\n",
                Tape->Chan[i]->Symbol.Mod_Name,
                Tape->Chan[i]->Symbol.Par_Name );
    fprintf(fp, "end\n");
    return;
}

ls_record()
{
    static int  inited = 0;
    SCALAR      value;
    static short interp;
    int i, result;

    if (inited == 0)
    {
        inited = -1;

        interp = INTERP;
        ls_record_channels_init();      /* make sure we've set up the Tape header &
channels */
        ls_record_alloc_storage();      /* then allocate data storage for each chann
el */

    } /* end of initialization section */



    if( !noTape )               /* Run time section */
      {                         /* Advance pointers */
        Tape->Current = Tape->Next;
        Tape->Last = Tape->Current;
        if (Tape->Current == Tape->First) Tape->First++;
        if (Tape->First >= Tape->Length) Tape->First = 0;
        if (Tape->First < 0) Tape->First = 0; /* To handle startup */

        Tape->T_Stamp[ Tape->Current ] = Simtime;   /* save time stamp */

        for (i=0;i<Tape->Num_Chan;i++)
            if (Tape->Chan[i]->Symbol.Addr)
                {
                    value = ls_get_double( Tape->Chan[i]->Symbol.Par_Type, Tape->Cha
n[i]->Symbol.Addr );
                    if ( !( value <= Tape->Chan[i]->Max_value ) ) Tape->Chan[i]->Max
_value = value;
                    if ( !( value >= Tape->Chan[i]->Min_value ) ) Tape->Chan[i]->Min
_value = value;
                    Tape->Chan[i]->Data[ Tape->Current ] = value;
                } /* end of for(i=0;i<Tape->Num_Chan;i++) loop */

        /* Only advance Next pointer if this is a 'keeper' frame */

        if (interp == INTERP) Tape->Next++;
        if (Tape->Next >= Tape->Length) Tape->Next = 0;
        if (--interp <= 0) interp = INTERP;
      } /* end of run time section */
  }
```

```
/***************************************************************************

        TITLE:          ls_settings.c

----------------------------------------------------------------------------

        FUNCTION:       Performs settings file utilities for LaRCsim

                Two major routines are provided in this module:
        ls_get_settings() and ls_put_settings(). These routines read and
        write the .set file used to record various LaRCsim user defined
        settings (run time length, trim variables, initial conditions,
        and variables to record, for example).

                The ls_get_settings() routine locates and opens (using
        ls_fopen()), also provided in this module) the appropriate settings
        file, and parses the information contained therein.  It has a list
        of facilities (such as IC, record, trim, etc.) that utilize variables
        in the settings file and calls their "get_set" routine as their
        keywords are encountered.

                The ls_fopen() routine searches for an appropriately named
        file somewhere along the LARCSIMPATH or within the default directory
        if no path is defined.

                The ls_put_settings() routine creates a new settings file
        and calls each facility's "put_set" routine to plop in the appropriate
        settings.

----------------------------------------------------------------------------

        MODULE STATUS:  incomplete

----------------------------------------------------------------------------

        GENEALOGY:      Created 950301 by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        950308  Added comment line with time stamp to ls_put_settings().
                                                                EBJ
        950314  Added init facility.                            EBJ
        950406  Added #includes of ls_types.h & ls_tape.h (to gain access
                to Tape->Length parameter); added new facility:
                ls_sim_get_set() and ls_sim_put_set(), to save and
                restore simulation option settings.             EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_settings.c,v 1.6 1995/04/07 01:35:58 bjax Exp $
$Log: ls_settings.c,v $
 * Revision 1.6  1995/04/07  01:35:58  bjax
 * Added ls_sim_get_set() and ls_sim_put_set() routines to save & restore
 * simulation options.
 *
 * Revision 1.5  1995/03/15  12:22:31  bjax
 * Added init facility; reworked logic of ls_get_settings() so that
 * a file name can be passed; if no file name is supplied, the default
 * settings file is opened.
 * EBJ
 *
 * Revision 1.4  1995/03/08  12:30:42  bjax
 * Added time, date, and user stamp to comment line in settings file output.
 *
 * Revision 1.3  1995/03/07  22:34:26  bjax
 * Added guts to ls_put_settings(); now have two facilities online: trim & record.
 *
 * Revision 1.2  1995/03/06  18:47:15  bjax
 * Reworked the facility list so that "set" facilities are passed the
 * pointer to the end of the buffer, so they can detect overruns, and
 * return pointer to next token in buffer. EBJ
 *
 * Revision 1.1  1995/03/03  02:17:34  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#include <limits.h>     /* defines PATH_MAX */
#include <sys/types.h>  /* needed for stat(3C) */
#include <sys/stat.h>
#include <sys/param.h>  /* needed for realpath(3C) */
#include <stdlib.h>     /* needed for realpath(3C), getenv(3C) */
#include <stdio.h>
#include <string.h>
#include "ls_types.h"
#include "ls_constants.h"       /* for NIL_POINTER and PATHNAME */
#include "ls_sim_control.h"     /* for simname */
#include "ls_tape.h"            /* for TAPE def'n */

extern TAPE *Tape;

static char rcsid[] = "$Id: ls_settings.c,v 1.6 1995/04/07 01:35:58 bjax Exp $";

#define DEFAULT_PATH "./"
#define PATH_SEP  ":"
#define MAX_PATHNAME_LENGTH PATH_MAX
#define MAX_LINE_SIZE 255
```

```c
#define INDIRECT_FLAG '@'
#define COMMENT_FLAG '#'

FILE *ls_fopen(const char *filename, const char *type, struct stat *statbuf )
{
    static int init = 0;
    static char *pathname;

    FILE *fp = 0L;
    char *tempname, *homedir;
    char qual_name_buf[ MAX_PATHNAME_LENGTH ];
    int status;
    int len;

    if (init == 0)

        /* Initializes the static string pathname to contain either
           the current directory path "./" or the pathname specified
           in an environment pathname (defined in ls_constants.h) */
        {
            init = -1;

            tempname = getenv( PATHNAME );
            if (tempname == NIL_POINTER)
                {
                    pathname = (char *) malloc(strlen(DEFAULT_PATH));
                    strcpy(pathname, DEFAULT_PATH);
                }
            else
                {
                    pathname = (char *) malloc(strlen(tempname));
                    strcpy(pathname, tempname);
                }
        }

    /* Now try to construct the fully qualified file name and open it.
       If the filename starts with a slash, ignore the pathname and
       try to open the absolute filename provided */

    if ((filename[0] == '/') || (*type == 'w'))  /* no path allowed */
        {
            return fopen( filename, type );
        }
    else
        /* Here if filename isn't absolute. Cycle through each of the
           directories listed in pathname, expanding a leading ~ into
           the user's home directory, and any ./ or ../ into default
           or previous directory, until either a file is located or the
           directory list is exhausted. */
        {
            tempname = strtok(pathname, PATH_SEP);  /* tempname has first dir */
            do
                {
                    len = strlen(tempname);
                    if (len > MAX_PATHNAME_LENGTH) break;
                    if (len <= 0) break;         /* shouldn't happen */
                    if (tempname[0] == '~')
                                /* expand home directory name */
                        {
                            homedir = getenv("HOME");
                            if (homedir == NIL_POINTER) break; /* give up */
                            len = strlen( homedir );
                            if (len > MAX_PATHNAME_LENGTH) break;
                            if (len <= 0) break;
                            strcpy( qual_name_buf, homedir );
                            strncat( qual_name_buf, tempname+1,
                                    MAX_PATHNAME_LENGTH-strlen(qual_name_buf));
                            tempname = qual_name_buf; /* point to new str */
                        }
                    tempname = realpath( tempname, qual_name_buf );
                                /* expand ./ or ../ directory name */
                    if (tempname == NIL_POINTER) break;
                    if (tempname[len-1] != '/')
                        strcat(qual_name_buf, "/" );    /* add final slash */
                    tempname =                  /* append file name */
                        strncat(qual_name_buf, filename,
                                MAX_PATHNAME_LENGTH-strlen(qual_name_buf));
                    if (tempname)
                        {
                            /* if still a valid string, check to see if
                               'tempname' describes an existing file */
                            status = stat( qual_name_buf, statbuf );
                            if (!status)
                                {
                                    return fopen( tempname, type );
                                }
                            tempname = strtok(NIL_POINTER, PATH_SEP);
                        }
                    else break;
                }
            while (tempname);
        }

    return fp;

}


#define NUM_FACILITIES 4
struct fac
{
    char *keyword;
    char *(*get_set_func)(); /* pointers to functions returning ptr to char */
    void  (*put_set_func)(); /* pointers to functions returning void */
};

static struct fac facilities[NUM_FACILITIES];

        char *ls_sim_get_set(    char *buffer, char *eob );
extern  char *ls_record_get_set( char *buffer, char *eob );
extern  char *ls_trim_get_set(   char *buffer, char *eob );
extern  char *ls_init_get_set(   char *buffer, char *eob );

       void ls_sim_put_set(   FILE *fp );
extern void ls_record_put_set(FILE *fp );
extern void ls_trim_put_set(  FILE *fp );
extern void ls_init_put_set(  FILE *fp );


void init_fac_list()
/* Initialize the above facility list */
{
    static char keyword0[] = "sim";
    static char keyword1[] = "record";
    static char keyword2[] = "trim";
    static char keyword3[] = "init";

    facilities[0].keyword = keyword0;
    facilities[1].keyword = keyword1;
    facilities[2].keyword = keyword2;
```

```
    facilities[3].keyword = keyword3;

    facilities[0].get_set_func = &ls_sim_get_set;
    facilities[1].get_set_func = &ls_record_get_set;
    facilities[2].get_set_func = &ls_trim_get_set;
    facilities[3].get_set_func = &ls_init_get_set;

    facilities[0].put_set_func = &ls_sim_put_set;
    facilities[1].put_set_func = &ls_record_put_set;
    facilities[2].put_set_func = &ls_trim_put_set;
    facilities[3].put_set_func = &ls_init_put_set;

}


int ls_parse_settings(FILE *fp, struct stat *statbuf )
    /* this routine reads the settings file into a buffer,
       resolves comment lines & includes, and calls the
       appropriate get_set routines.    */
{
    char *buffer, *bufptr, *endptr, **lasts, *nullptr, null = '\0';
    char line[MAX_LINE_SIZE];
    off_t filesize;
    int status, n, i, skiptok;
    FILE *new_settings_file;
    struct stat *new_statbuf;

    nullptr = &null;
    lasts = &nullptr;

    /* read file into buffer */

    filesize = statbuf->st_size;
    buffer = (char *) malloc(filesize);
    bufptr = buffer;
    endptr = buffer+filesize;   /* points to one char past end of buffer */

    if (buffer == NIL_POINTER) return -1;
    status = fread(buffer, filesize, 1, fp);
    if (status != 1) { free( buffer ); return -1; }

    bufptr = strtok_r( buffer, "\n", lasts);
    while ((bufptr < endptr) && (bufptr >= buffer))
        {
            skiptok = 0;
            n = sscanf(bufptr, "%s", line);
            switch( line[0] )       /* examine first char */
                {
                case INDIRECT_FLAG:
                    /* if the first character of a line starts with redirection
                       symbol, open that file and parse it */

                    new_statbuf = (struct stat *)malloc(sizeof( struct stat ));
                    if (new_statbuf == NIL_POINTER) {free(buffer); return -1;}
                    new_settings_file = ls_fopen( bufptr+1, "r", new_statbuf);
                    if (new_settings_file)
                        {
                            status = ls_parse_settings( new_settings_file,
                                                        new_statbuf );
                            if (status) fprintf(stderr,
                                "Error parsing subordinate settings file.\n");
                            fclose(new_settings_file);
                        }
                    free( new_statbuf );
                    break;
                case COMMENT_FLAG:
                    /* skip over comments */
                    break;
                default:        /* not a comment or redirection; look at word */
                    for(i=0;i<NUM_FACILITIES;i++)
                        if (strcasecmp( line, facilities[i].keyword ) == 0)
                            {
/*                              printf("%s\n", facilities[i].keyword); */
                                bufptr =
                                  (*facilities[i].get_set_func)(bufptr,endptr);
                                        /* call the command that matches */
                                skiptok = 1;
                                break;
                            }
                }
            if (skiptok)
                {
                    lasts = &nullptr; /* reset strtok search */
                    bufptr = strtok_r(bufptr, "\n", lasts);
                }
            else
                bufptr = strtok_r( NIL_POINTER, "\n", lasts );
        }

    free( buffer );
    return 0;
}

void ls_get_settings( char *desired_file_name )
{
    FILE *settings_file;
    struct stat statbuf;
    int status, namelen;

    char *settings_file_name;

    if (desired_file_name[0] == '\0')
        {
            settings_file_name = (char *) malloc( strlen(sim_control_.simname) + 1 )
;
            if (settings_file_name)
                {
                    settings_file_name[0] = '.';
                    settings_file_name[1] = '\0';
                    strcat(settings_file_name, sim_control_.simname);
                }
        }
    else
        {  /* make local copy */
            namelen = strlen( desired_file_name )+1;
            settings_file_name = (char *) malloc( namelen );
            if (settings_file_name) strncpy( settings_file_name, desired_file_name,
namelen );
        }


    if (settings_file_name)
        {
            settings_file = ls_fopen( settings_file_name, "r", &statbuf );
            if (settings_file)
                {
                    init_fac_list();
                    status = ls_parse_settings( settings_file, &statbuf );
                    if (status) fprintf(stderr, "Error parsing settings file.\n");
                    fclose( settings_file );
```

```c
            }
        }
    free( settings_file_name );
}

void ls_put_settings()
{
    FILE *settings_file;
    char *settings_file_name;
    struct stat statbuf;
    int i;

    settings_file_name = (char *) malloc( strlen(sim_control_.simname) + 3 );
    if (settings_file_name)
        {
            settings_file_name[0] = '.';
            settings_file_name[1] = '/';
            settings_file_name[2] = '.';
            settings_file_name[3] = '\0';
            strcat(settings_file_name, sim_control_.simname);

            settings_file = ls_fopen( settings_file_name, "w", &statbuf );
            if (settings_file)
                {
                    init_fac_list();
                    fprintf(settings_file, "# .%s created at %s %s by %s\n",
                            sim_control_.simname,
                            sim_control_.date_string,
                            sim_control_.time_stamp,
                            sim_control_.userid );
                    for(i=0;i<NUM_FACILITIES;i++)
                            (*facilities[i].put_set_func)(settings_file);
                    (void) fclose( settings_file );
                }
        }
    free( settings_file_name );
}

#define FACILITY_NAME_STRING "sim"
#define CURRENT_VERSION 10

void ls_sim_put_set( FILE *fp )
{
    int i;

    if (fp==0) return;
    fprintf(fp, "\n");
    fprintf(fp, "#==============================  %s\n", FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, "%04d\n", CURRENT_VERSION);
    fprintf(fp, "    write_av       %d\n", sim_control_.write_av);
    fprintf(fp, "    write_mat      %d\n", sim_control_.write_mat);
    fprintf(fp, "    write_tab      %d\n", sim_control_.write_tab);
    fprintf(fp, "    write_ascl     %d\n", sim_control_.write_ascl);
    fprintf(fp, "    write_spacing  %d\n", sim_control_.write_spacing);
    fprintf(fp, "    end_time       %f\n", sim_control_.end_time);
    fprintf(fp, "    model_hz       %f\n", sim_control_.model_hz);
    fprintf(fp, "    term_update_hz %f\n", sim_control_.term_update_hz);
    fprintf(fp, "    data_rate      %f\n", sim_control_.model_hz / sim_control_.save_spa
cing);
    fprintf(fp, "    buffer_time    %f\n", Tape->Length*sim_control_.save_spacing/sim_co
ntrol_.model_hz);
    fprintf(fp, "end\n");

    return;
}


char *ls_sim_get_set(char *buffer, char *eob)
/* This routine parses the settings file for "sim" entries. */
{

    static char *fac_name = FACILITY_NAME_STRING;
    char *bufptr, **lasts, *nullptr, null = '\0';
    int n, ver, i, error;
    float buffer_time, data_rate;
    char line[MAX_LINE_SIZE];
    char word[MAX_LINE_SIZE];
    char value[MAX_LINE_SIZE];

    /* set default values */

    buffer_time = sim_control_.time_slices * sim_control_.save_spacing / sim_control
_.model_hz;
    data_rate   = sim_control_.model_hz / sim_control_.save_spacing;

    nullptr = &null;
    lasts   = &nullptr;

    n = sscanf(buffer, "%s", line);
    if (n == 0) return 0L;
    if (strncasecmp( fac_name, line, strlen(fac_name) )) return 0L;

    bufptr = strtok_r( buffer+strlen(fac_name)+1, "\n", lasts);
    if (bufptr == 0) return 0L;

    sscanf( bufptr, "%d", &ver );
    if (ver != CURRENT_VERSION) return 0L;

    while( eob > bufptr )
      {
        bufptr = strtok_r( 0L, "\n", lasts );
        if (bufptr == 0) return 0L;
        if (strncasecmp( bufptr, "end", 3) == 0) break;

        sscanf( bufptr, "%s", line );
        if (line[0] != '#') /* ignore comments */
          {
            n = sscanf( bufptr, "%s %s", word, value);
            if (n == 2)
              {
                    if (strncmp(word, "write_av"      ,  8) == 0) sim_control_.write
_av = atoi(value);
                    if (strncmp(word, "write_mat"     ,  9) == 0) sim_control_.write
_mat = atoi(value);
                    if (strncmp(word, "write_tab"     ,  9) == 0) sim_control_.write
_tab = atoi(value);
                    if (strncmp(word, "write_ascl"    , 10) == 0) sim_control_.write
_ascl = atoi(value);
                    if (strncmp(word, "write_spacing" , 13) == 0) sim_control_.write
_spacing = atoi(value);
                    if (strncmp(word, "end_time"      ,  8) == 0) sim_control_.end_t
ime = atof(value);
                    if (strncmp(word, "model_hz"      ,  8) == 0) sim_control_.model
_hz = atof(value);
                    if (strncmp(word, "term_update_hz", 14) == 0) sim_control_.term_
update_hz = atof(value);
                    if (strncmp(word, "data_rate"     ,  9) == 0) data_rate = atof(v
alue);
```

```
                    if (strncmp(word, "buffer_time"   , 11) == 0) buffer_time = atof(val
ue);
                }

            }
        }

    sim_control_.save_spacing = (int) (0.5 + sim_control_.model_hz / data_rate);
    if (sim_control_.save_spacing < 1) sim_control_.save_spacing = 1;

    sim_control_.time_slices = buffer_time * sim_control_.model_hz / sim_control_.save_s
pacing;
    if (sim_control_.time_slices < 2) sim_control_.time_slices = 2;

    bufptr = *lasts;
    return bufptr;
}
```

```
/***************************************************************************

        TITLE:  ls_step

----------------------------------------------------------------------------

        FUNCTION:       Integration routine for equations of motion
                        (vehicle states)

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Written 920802 by Bruce Jackson.  Based upon equations
                        given in reference [1] and a Matrix-X/System Build block
                        diagram model of equations of motion coded by David Raney
                        at NASA-Langley in June of 1992.

----------------------------------------------------------------------------

        DESIGNED BY:    Bruce Jackson

        CODED BY:       Bruce Jackson

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        921223  Modified calculation of Phi and Psi to use the "atan2" routine
                rather than the "atan" to allow full circular angles.
                "atan" limits to +/- pi/2.                              EBJ

        940111  Changed from oldstyle include file ls_eom.h; also changed
                from DATA to SCALAR type.                               EBJ

        950207  Initialized Alpha_dot and Beta_dot to zero on first pass; calculated
                thereafter.                                             EBJ

        950224  Added logic to avoid adding additional increment to V_east
                in case V_east already accounts for rotating earth.
                                                                        EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_step.c,v 1.5 1995/03/02 20:24:13 bjax Stab $
$Log: ls_step.c,v $
 * Revision 1.5  1995/03/02  20:24:13  bjax
 * Added logic to avoid adding additional increment to V_east
 * in case V_east already accounts for rotating earth. EBJ
 *
 * Revision 1.4  1995/02/07  20:52:21  bjax
 * Added initialization of Alpha_dot and Beta_dot to zero on first
 * pass; they get calculated by ls_aux on next pass...  EBJ
 *
 * Revision 1.3  1994/01/11  19:01:12  bjax
 * Changed from DATA to SCALAR type; also fixed header files (was ls_eom.h)
 *
 * Revision 1.2  1993/06/02  15:03:09  bjax
 * Moved initialization of geocentric position to subroutine ls_geod_to_geoc.
 *
 * Revision 1.1  92/12/30  13:16:11  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

                [ 1]    McFarland, Richard E.: "A Standard Kinematic Model
                        for Flight Simulation at NASA-Ames", NASA CR-2497,
                        January 1975

                [ 2]    ANSI/AIAA R-004-1992 "Recommended Practice: Atmos-
                        pheric and Space Flight Vehicle Coordinate Systems",
                        February 1992


----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:       None.

----------------------------------------------------------------------------

        INPUTS: State derivatives

----------------------------------------------------------------------------

        OUTPUTS:        States

--------------------------------------------------------------------------*/

#include "ls_types.h"
#include "ls_constants.h"
#include "ls_generic.h"
#include "ls_sim_control.h"
#include <math.h>

extern SCALAR Simtime;          /* defined in ls_main.c */

void ls_step( dt, Initialize )

SCALAR dt;
int Initialize;

{
        static  int     inited = 0;
                SCALAR  dth;
        static  SCALAR  v_dot_north_past, v_dot_east_past, v_dot_down_past;
        static  SCALAR  latitude_dot_past, longitude_dot_past, radius_dot_past;
        static  SCALAR  p_dot_body_past, q_dot_body_past, r_dot_body_past;
                SCALAR  p_local_in_body, q_local_in_body, r_local_in_body;
                SCALAR  epsilon, inv_eps, local_gnd_veast;
                SCALAR  e_dot_0, e_dot_1, e_dot_2, e_dot_3;
        static  SCALAR  e_0, e_1, e_2, e_3;
        static  SCALAR  e_dot_0_past, e_dot_1_past, e_dot_2_past, e_dot_3_past;

/*  I N I T I A L I Z A T I O N   */


        if ( (inited == 0) || (Initialize != 0) )
```

```
	{
/* Set past values to zero */
	v_dot_north_past = v_dot_east_past = v_dot_down_past      = 0;
	latitude_dot_past = longitude_dot_past = radius_dot_past  = 0;
	p_dot_body_past = q_dot_body_past = r_dot_body_past       = 0;
	e_dot_0_past = e_dot_1_past = e_dot_2_past = e_dot_3_past = 0;

/* Initialize geocentric position from geodetic latitude and altitude */

	ls_geod_to_geoc( Latitude, Altitude, &Sea_level_radius, &Lat_geocentric);
	Earth_position_angle = 0;
	Lon_geocentric = Longitude;
	Radius_to_vehicle = Altitude + Sea_level_radius;

/* Correct eastward velocity to account for earths' rotation, if necessary */

	local_gnd_veast = OMEGA_EARTH*Sea_level_radius*cos(Lat_geocentric);
	if( fabs(V_east - V_east_rel_ground) < 0.8*local_gnd_veast )
	V_east = V_east + local_gnd_veast;

/* Initialize quaternions and transformation matrix from Euler angles */

	    e_0 = cos(Psi*0.5)*cos(Theta*0.5)*cos(Phi*0.5)
		+ sin(Psi*0.5)*sin(Theta*0.5)*sin(Phi*0.5);
	    e_1 = cos(Psi*0.5)*cos(Theta*0.5)*sin(Phi*0.5)
		- sin(Psi*0.5)*sin(Theta*0.5)*cos(Phi*0.5);
	    e_2 = cos(Psi*0.5)*sin(Theta*0.5)*cos(Phi*0.5)
		+ sin(Psi*0.5)*cos(Theta*0.5)*sin(Phi*0.5);
	    e_3 =-cos(Psi*0.5)*sin(Theta*0.5)*sin(Phi*0.5)
		+ sin(Psi*0.5)*cos(Theta*0.5)*cos(Phi*0.5);
	    T_local_to_body_11 = e_0*e_0 + e_1*e_1 - e_2*e_2 - e_3*e_3;
	    T_local_to_body_12 = 2*(e_1*e_2 + e_0*e_3);
	    T_local_to_body_13 = 2*(e_1*e_3 - e_0*e_2);
	    T_local_to_body_21 = 2*(e_1*e_2 - e_0*e_3);
	    T_local_to_body_22 = e_0*e_0 - e_1*e_1 + e_2*e_2 - e_3*e_3;
	    T_local_to_body_23 = 2*(e_2*e_3 + e_0*e_1);
	    T_local_to_body_31 = 2*(e_1*e_3 + e_0*e_2);
	    T_local_to_body_32 = 2*(e_2*e_3 - e_0*e_1);
	    T_local_to_body_33 = e_0*e_0 - e_1*e_1 - e_2*e_2 + e_3*e_3;

/*	Calculate local gravitation acceleration	*/

		ls_gravity( Radius_to_vehicle, Lat_geocentric, &Gravity );

/*	Initialize vehicle model			*/

		ls_aux();
		ls_model();

/*	Calculate initial accelerations */

		ls_accel();

/* Initialize auxiliary variables */

		ls_aux();
		Alpha_dot = 0.;
		Beta_dot = 0.;

/* set flag; disable integrators */

		inited = -1;
		dt = 0;

	}
```



```
/* Update time */

	dth = 0.5*dt;
	Simtime = Simtime + dt;

/*  L I N E A R   V E L O C I T I E S   */

/* Integrate linear accelerations to get velocities */
/*    Using predictive Adams-Bashford algorithm     */

    V_north = V_north + dth*(3*V_dot_north - v_dot_north_past);
    V_east  = V_east  + dth*(3*V_dot_east  - v_dot_east_past );
    V_down  = V_down  + dth*(3*V_dot_down  - v_dot_down_past );

/* record past states */

    v_dot_north_past = V_dot_north;
    v_dot_east_past  = V_dot_east;
    v_dot_down_past  = V_dot_down;

/* Calculate trajectory rate (geocentric coordinates) */

    if (cos(Lat_geocentric) != 0)
	Longitude_dot = V_east/(Radius_to_vehicle*cos(Lat_geocentric));

    Latitude_dot = V_north/Radius_to_vehicle;
    Radius_dot = -V_down;

/*  A N G U L A R   V E L O C I T I E S   A N D   P O S I T I O N S */

/* Integrate rotational accelerations to get velocities */

    P_body = P_body + dth*(3*P_dot_body - p_dot_body_past);
    Q_body = Q_body + dth*(3*Q_dot_body - q_dot_body_past);
    R_body = R_body + dth*(3*R_dot_body - r_dot_body_past);

/* Save past states */

    p_dot_body_past = P_dot_body;
    q_dot_body_past = Q_dot_body;
    r_dot_body_past = R_dot_body;

/* Calculate local axis frame rates due to travel over curved earth */

    P_local =  V_east/Radius_to_vehicle;
    Q_local = -V_north/Radius_to_vehicle;
    R_local = -V_east*tan(Lat_geocentric)/Radius_to_vehicle;

/* Transform local axis frame rates to body axis rates */

    p_local_in_body = T_local_to_body_11*P_local + T_local_to_body_12*Q_local + T_lo
cal_to_body_13*R_local;
    q_local_in_body = T_local_to_body_21*P_local + T_local_to_body_22*Q_local + T_lo
cal_to_body_23*R_local;
    r_local_in_body = T_local_to_body_31*P_local + T_local_to_body_32*Q_local + T_lo
cal_to_body_33*R_local;

/* Calculate total angular rates in body axis */

    P_total = P_body - p_local_in_body;
    Q_total = Q_body - q_local_in_body;
    R_total = R_body - r_local_in_body;

/* Transform to quaternion rates (see Appendix E in [2]) */
```

```
    e_dot_0 = 0.5*( -P_total*e_1 - Q_total*e_2 - R_total*e_3 );
    e_dot_1 = 0.5*(  P_total*e_0 - Q_total*e_3 + R_total*e_2 );
    e_dot_2 = 0.5*(  P_total*e_3 + Q_total*e_0 - R_total*e_1 );
    e_dot_3 = 0.5*( -P_total*e_2 + Q_total*e_1 + R_total*e_0 );

/* Integrate using trapezoidal as before */

        e_0 = e_0 + dth*(e_dot_0 + e_dot_0_past);
        e_1 = e_1 + dth*(e_dot_1 + e_dot_1_past);
        e_2 = e_2 + dth*(e_dot_2 + e_dot_2_past);
        e_3 = e_3 + dth*(e_dot_3 + e_dot_3_past);

/* calculate orthagonality correction  - scale quaternion to unity length */

        epsilon = sqrt(e_0*e_0 + e_1*e_1 + e_2*e_2 + e_3*e_3);
        inv_eps = 1/epsilon;

        e_0 = inv_eps*e_0;
        e_1 = inv_eps*e_1;
        e_2 = inv_eps*e_2;
        e_3 = inv_eps*e_3;

/* Save past values */

        e_dot_0_past = e_dot_0;
        e_dot_1_past = e_dot_1;
        e_dot_2_past = e_dot_2;
        e_dot_3_past = e_dot_3;

/* Update local to body transformation matrix */

        T_local_to_body_11 = e_0*e_0 + e_1*e_1 - e_2*e_2 - e_3*e_3;
        T_local_to_body_12 = 2*(e_1*e_2 + e_0*e_3);
        T_local_to_body_13 = 2*(e_1*e_3 - e_0*e_2);
        T_local_to_body_21 = 2*(e_1*e_2 - e_0*e_3);
        T_local_to_body_22 = e_0*e_0 - e_1*e_1 + e_2*e_2 - e_3*e_3;
        T_local_to_body_23 = 2*(e_2*e_3 + e_0*e_1);
        T_local_to_body_31 = 2*(e_1*e_3 + e_0*e_2);
        T_local_to_body_32 = 2*(e_2*e_3 - e_0*e_1);
        T_local_to_body_33 = e_0*e_0 - e_1*e_1 - e_2*e_2 + e_3*e_3;

/* Calculate Euler angles */

        Theta = asin( -T_local_to_body_13 );

        if( T_local_to_body_11 == 0 )
        Psi = 0;
        else
        Psi = atan2( T_local_to_body_12, T_local_to_body_11 );

        if( T_local_to_body_33 == 0 )
        Phi = 0;
        else
        Phi = atan2( T_local_to_body_23, T_local_to_body_33 );

/* Resolve Psi to 0 - 359.9999 */

        if (Psi < 0 ) Psi = Psi + 2*PI;

/*  L I N E A R   P O S I T I O N S   */

/* Trapezoidal acceleration for position */

        Lat_geocentric    = Lat_geocentric    + dth*(Latitude_dot  + latitude_dot_past )
;
        Lon_geocentric    = Lon_geocentric    + dth*(Longitude_dot + longitude_dot_p
ast);
        Radius_to_vehicle = Radius_to_vehicle + dth*(Radius_dot    + radius_dot_past
 );
        Earth_position_angle = Earth_position_angle + dt*OMEGA_EARTH;

/* Save past values */

        latitude_dot_past  = Latitude_dot;
        longitude_dot_past = Longitude_dot;
        radius_dot_past    = Radius_dot;

/* end of ls_step */
}
/***********************************************************************/
```

```
/***************************************************************************

        TITLE:          ls_sym

----------------------------------------------------------------------------

        FUNCTION:       Symbol table support routines for LaRCsim

----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created 930629 by E. B. Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    E. B. Jackson

        CODED BY:       ditto

        MAINTAINED BY:

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        940112  Restructured this routine to be more readable;
                added support for structure elements                    EBJ

        940505  Modified to use ELF structures and functions;
                had to rewrite most symbol table routines as interim
                between COFF and ELF, since older "ldfcn" routines
                will not be supported after IRIX 5.2                    EBJ

        950306  Added routines ls_get_sym_val() and ls_set_sym_val()

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_sym.c,v 2.7 1995/03/06 18:44:07 bjax Stab $
$Log: ls_sym.c,v $
 * Revision 2.7  1995/03/06  18:44:07  bjax
 * Added ls_get_sym_val and ls_set_sym_val() routines.
 *
 * Revision 2.6  1995/02/27  19:54:51  bjax
 * Added utility routines: ls_print_findsym_error(), ls_get_double(),
 * ls_set_double(). EBJ
 *
 * Revision 2.5  1994/05/17  15:07:40  bjax
 * Corrected so that full name to directory and file is used
 * to open symbol table, so that sims can be run from another
 * default directory.
 *
 * Revision 2.4  1994/05/11  16:25:29  bjax
 * Correct problem with bounds error checking on dimensioned variables
 * that were Typedefs.  Increased the allowable number of dimensions
 * to six from three. EBJ
 *
 * Revision 2.3  1994/05/06  20:19:30  bjax
 * More or less complete set of data types now supported.
 *
 * Revision 1.2  1993/07/30  17:37:42  bjax
 * Corrected logic to skip over unwanted procedures.  EBJ
 *
 * Revision 1.1  1993/07/27  23:43:21  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/
#include <libelf.h>
#include <syms.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include "ls_sym.h"

        /* local definitions */

#define FAILURE 0
#define SUCCESS -1
#define FALSE 0
#define TRUE -1

        /* macro functions to make code a little easier to read */

#define ISTRUCT(x) ((x->ti.bt == btStruct) || (x->ti.bt == btUnion))
#define ISTYPEDEF(x) (AUX( x.itaux )->ti.bt == btTypedef)
#define SYMEXIT(x) (free(tokenbuf); if(!index_list) free(index_list); return x;)
#define LOCALSYM(x) ((SYMR *) (hdrr_base + hdrr->cbSymOffset) + x)
#define EXTRNSYM(x) ((EXTR *) (hdrr_base + hdrr->cbExtOffset) + x)
#define FD(x)       ((FDR *)  (hdrr_base + hdrr->cbFdOffset ) + x)
#define AUX(x)      ((AUXU *) (hdrr_base + hdrr->cbAuxOffset) + x)
#define SS(x)   ((char *)(hdrr_base+hdrr->cbSsOffset)+FD(x.ifd)->issBase+x.psymr->iss
)
#define ESS(x) ((char *)(hdrr_base+hdrr->cbSsExtOffset)+x.psymr->iss)

        /*
         * special data structure typedef -- it combines
         * most information about each symbol.
         */

typedef struct
{
    short ext_sym;      /* flag to indicate it is located in ext syms */
    short ifd;          /* file descriptor index */
```

```
    long  isym;         /* symbol index */
    long  idaux;        /* abs. index to aux with array dims; else 0 */
    long  itaux;        /* abs. index into aux space for type aux    */
    pSYMR psymr;        /* local ptr to symbol */
} lsSYM;

        /* the following variable has Global scope */

extern char *fullname;

        /* the following variables have File scope */

static long     symmax, symmaxlocal, symmaxextern;
static long     i, end_of_proc;
static int      module_found, symbol_found;
static unsigned long hdrr_base; /* diff between section addr and offsets */
static HDRR     *hdrr;
static char     *namep;
static lsSYM    symbol;
```

```
/*
 *              ===== symtbread =====
 *
 * Given the absolute index of a symbol, this routine populates
 * the fields of the lsSYM data structure pointed to by the second
 * argument based on debugger symbol information
 */

static int symtbread( long j, lsSYM *mySym )

{
    pEXTR ext;
    FDR   *fd;
    short i;

    /* return FAILURE if out of bounds */

    if (j < 0) return FAILURE;
    if (j > symmaxextern + symmaxlocal) return FAILURE;

    /* point to proper symbol */

    mySym->isym = j;
    if (j <= symmaxlocal)
        {                                               /* local symbol */
            mySym->ext_sym = FALSE;
            mySym->psymr = LOCALSYM(j);
            for ( i = 0; i < hdrr->ifdMax; i++)
                {
                    fd = FD(i);
                    if (fd->isymBase > j) break;
                }
            mySym->ifd = i-1;
        }
    else                                                /* global symbol */
        {
            mySym->ext_sym = TRUE;
            ext = EXTRNSYM(j - symmaxlocal);
            mySym->ifd = ext->ifd;
            mySym->psymr = &ext->asym;
        }

    mySym->itaux = FD(mySym->ifd)->iauxBase + mySym->psymr->index;
    mySym->idaux = 0;

    return SUCCESS;

}       /* end of symtbread */
```

```
/*
 *              ===== symgetname =====
 *
 *
 * Given an lsSYM symbol record, this routine returns a pointer
 * to a string containing the name of the symbol.
 */

static char *symgetname( lsSYM sym )
{
    if (sym.ext_sym == 0)
        return SS(sym);
    else
        return ESS(sym);
}

/*
 *              ===== loadSymbols =====
 *
 * This routine reads in the debugger symbol table section from the
 * executable file pointed to by global string progname. It returns
 * SYM_OPEN_ERR if the file can't be opened, or SYM_NO_SYMS if it
 * doesn't appear to have debugging information. It also sets the file
 * scope variables symmaxlocal and symmaxextern and calculates the
 * appropriate value of hdrr_base.
 */

static int loadSymbols( void )  /* open and read the symbol table */
{
    int         fildes;
    Elf         *elf;           /* ELF file pointer     */
    Elf_Scn     *scn;           /* ELF section          */
    Elf32_Shdr  *shdr;          /* ELF section header   */
    Elf_Data    *data;          /* ELF data member      */

    if (elf_version(EV_CURRENT) == EV_NONE) return SYM_OPEN_ERR;
    fildes = open(fullname, O_RDONLY);
    if ((elf = elf_begin(fildes, ELF_C_READ, (Elf *)0)) == 0)
        return SYM_OPEN_ERR;
    scn = (Elf_Scn *)0;
    do
      {
        if ((scn = elf_nextscn( elf, scn )) == 0)
            return SYM_NO_SYMS;
        if ((shdr = elf32_getshdr(scn)) == 0)
            return SYM_NO_SYMS;
      }
    while (shdr->sh_type != SHT_MIPS_DEBUG);    /* special MIPS section */

    data = (Elf_Data *)0;
    if (    ((data = elf_getdata( scn, data )) == 0 )
        || ( data->d_size == 0 )
        || ( data->d_buf == 0 ) )
        return SYM_NO_SYMS;

    hdrr = (pHDRR) data->d_buf; /* save pointer to symbolic header */
    hdrr_base =  (off_t) hdrr - shdr->sh_offset;

/*    elf_end( elf );   /* free up elf descriptor -deleted 5/11 EBJ;
                         it appears to deallocate the sym table. */
    close( fildes );    /* close open file descriptor */

    symmaxlocal  = hdrr->isymMax;
    symmaxextern = hdrr->iextMax;

    return SYM_OK;
}
```

```
/*
 *              ===== lookForModule =====
 *
 * This routine searches the debugger symbols table, starting at
 * the index location pointed to by file variable i, for a symbol
 * whose associated string matches the string pointed to by modname.
 * If successful, it returns the absolute symbol index of the found
 * module.
 */

static int lookForModule( const char *modname )

{

 AUXU *paux;

 while( i < symmax)
  {
    if (FAILURE == symtbread(i, &symbol)) return SYM_UNEXPECTED_ERR;
    namep = symgetname( symbol );
    if ( namep == NULL ) return SYM_UNEXPECTED_ERR;

    if ( symbol.psymr->st == stProc )      /* beginning of a procedure */
      {
        paux = AUX( symbol.itaux ); /* get isymMac */
        if ( paux == auxNil ) return SYM_UNEXPECTED_ERR;

        end_of_proc = paux->isym + FD(symbol.ifd)->isymBase - 1;
        /* should point to one before next proc */

        module_found = !strcmp(namep, modname); /* returns 0 if equal */
        if( !module_found )
          {
            if ( i < symmaxlocal )
              i = end_of_proc;  /* skip around procedure for speed */
          }
        else
          return SYM_OK;        /* module found successfully */
      }
/*    else */

        /* here if module not found and symbol is not stProc type
           - do nothing; fall through and let counter increment */

    i++;        /* increment symbol index */
  }
  return SYM_MOD_NOT_FOUND;
}
```

```
/*
 *              ===== followRFD =====
 *
 *   This routine follows the rfd trail - that is, it takes file
 *   indirection information from the aux entries for a given symbol
 *   and returns a pointer to another symbol entry to which the original
 *   symbol refers - for example, a global symbol entry to a specific
 *   definition, or a typedef to its definition.
 *
 *   The argument i is an absolute index to a symbol contained
 *   in the symbol table; as is the return value (which represents the
 *   indirect referenced symbol).
 */


static long followRFD( long i )

{
  lsSYM symbol;
  AUXU  *paux;
  long  iss, rfdbase;
  short irfd, newifd;
  RFDT  *pfd;

  if (FAILURE == symtbread(i, &symbol))
      return 0;

  paux = AUX( symbol.itaux+1 );                   /* get iss */
  if ( paux->rndx.rfd != 4095 ) return 0; /* unexpected result */

  iss = paux->rndx.index;               /* and save it */

  paux = AUX( symbol.itaux+2 );                  /* get rfd index */
  irfd = paux->isym;  /* this seems undocumented, but works */


  if (irfd < 0 || irfd > FD(symbol.ifd)->crfd)
      return SYM_UNEXPECTED_ERR;        /* bounds check */

  rfdbase = FD(symbol.ifd)->rfdBase;    /* get base of rfd entries */

  pfd = (RFDT *) (hdrr_base + hdrr->cbRfdOffset) +
      rfdbase + irfd;                   /* point to new fd */

  newifd = *pfd;                        /* dereference pointer */

  return iss + FD(newifd)->isymBase;    /* and get new isym */

}
```

```
/*
 *              ===== lookForSym =====
 *
 *   This routine searches through symbol table, starting at present
 *   location pointed to by i, up to the end of the procedure (pointed
 *   to by end_of_proc), for a symbol whose string matches symname.
 *
 *   If not found, this routine returns SYM_VAR_NOT_FOUND. If the
 *   symbol is found, but isn't a static variable, this routine returns
 *   SYM_NOT_STATIC. If a static symbol is found that is NOT a structure,
 *   but expecting_struct is TRUE, this routine returns
 *   SYM_UNEXPECTED_ERR. If a static symbol is found that IS a structure,
 *   but a scalar was expected, this routine returns SYM_NOT_SCALAR. If a
 *   static symbol with the proper name is found within the procedure
 *   symbol space that is not a structure (and expecting_struct is FALSE),
 *   this routine returns SYM_OK with "i" pointing to the symbol entry and
 *   "addr" loaded with the value of that symbol's address.
 *   If a static symbol is found that is a structure, and expecting_struct
 *   is TRUE, this routine returns SYM_OK with "i" pointing to the
 *   structure's stBlock symbol table entry and "addr" loaded with the value
 *   of the structure's beginning address. Any other result should return
 *   SYM_UNEXPECTED_ERR.
 */

static int lookForSym( char *symname, int lookingForMember,
                int expecting_struct, char **addr, int num_indices )
{
  long firstSym, lastSym;
  long idaux, itaux;
  AUXU *daux, *taux;

  symbol_found = FALSE;
  lastSym = end_of_proc;

  if ( FAILURE == symtbread(i, &symbol)) return SYM_UNEXPECTED_ERR;
  if ( symbol.psymr->st == stStruct ) /* if we're looking in a structure... */
      {
        firstSym = i;                   /* save start of structure */
        i = symbol.psymr->index +
            FD(symbol.ifd)->isymBase - 1;      /* point to end of structure */

        if ( FAILURE == symtbread(i, &symbol) ) return SYM_UNEXPECTED_ERR;
        if ( symbol.psymr->st != stEnd ) return SYM_UNEXPECTED_ERR;

        lastSym = i;

        i = firstSym+1;
        if( (i < firstSym) || (i > lastSym) ) return SYM_UNEXPECTED_ERR;
    }

  while( i < symmax ) /* loop, but make absolutely certain not to go off end */
    {
      if ( FAILURE == symtbread(i, &symbol)) return SYM_UNEXPECTED_ERR;
      namep = symgetname( symbol );
      if ( namep == NULL ) return SYM_UNEXPECTED_ERR;
      if ( symbol_found = !strcmp( namep, symname ) )
        {
          /* symbol found  -- update address info */

          if ( !lookingForMember )  /* looking for static symbol */
            {
              if ( !( ( symbol.psymr->st == stStatic ) ||
                      ( symbol.psymr->st == stGlobal ) ) )
                  return SYM_NOT_STATIC;
              *addr = (char *) symbol.psymr->value;
            }
          else          /* looking for subsequent structure member */
            {
              if ( symbol.psymr->st != stMember ) return SYM_UNEXPECTED_ERR;
              *addr = *addr + symbol.psymr->value/8;
            }

          idaux = 0;
          daux = AUX( symbol.itaux );
          if ( daux == NULL ) return SYM_UNEXPECTED_ERR;

          if ( daux->ti.tq0 == tqArray ) /* array element found */
              {
                  if (num_indices <= 0) return SYM_NOT_SCALAR;
                  idaux = symbol.itaux; /* save pointer to dim. info */
              }

          while (ISTYPEDEF(symbol))      /* dereference to get base type */
            {
                i = followRFD( i );
                if ( i == 0 ) return SYM_UNEXPECTED_ERR;
                if ( FAILURE == symtbread(i, &symbol))
                    return SYM_UNEXPECTED_ERR;
                /* check for proper array-ness */

                daux = AUX( symbol.itaux );
                if ( daux == NULL ) return SYM_UNEXPECTED_ERR;

                if ( daux->ti.tq0 == tqArray ) /* array element found */
                    {
                        if (num_indices <= 0) return SYM_NOT_SCALAR;
                        idaux = symbol.itaux;   /* save pointer to dim. info */
                    }
            }

          if ((num_indices > 0) && (idaux == 0)) return SYM_UNEXPECTED_ERR;
          symbol.idaux = idaux;         /* restore pointer to array, if any */

          taux = AUX( symbol.itaux );  /* get type aux entry */
          if ( taux == NULL ) return SYM_UNEXPECTED_ERR;

          if (expecting_struct && !ISTRUCT(taux)) return SYM_UNEXPECTED_ERR;
          if (!expecting_struct && ISTRUCT(taux)) return SYM_NOT_SCALAR;

          if (ISTRUCT(taux))  /* need to point to stStruct sym */
            {
                if ( symbol.psymr->st == stGlobal ) /* need to find stStruct */
                    {
                        i = followRFD( i );
                        if ( i == 0 ) return SYM_UNEXPECTED_ERR;
                        if ( FAILURE == symtbread(i, &symbol))
                            return SYM_UNEXPECTED_ERR;
                        taux = AUX( symbol.itaux );
                        if ( taux == NULL ) return SYM_UNEXPECTED_ERR;
                    }
                if ( symbol.psymr->st == stStatic ||
                     symbol.psymr->st == stMember ) /* need stStruct */
                    {
                        i = i - 1;                  /* back up to stEnd */
                        if ( FAILURE == symtbread(i, &symbol))
                            return SYM_UNEXPECTED_ERR;
                        i = symbol.psymr->index +
                            FD(symbol.ifd)->isymBase; /* pt to stStruct */
                        if ( FAILURE == symtbread(i, &symbol))
```

```
                    return SYM_UNEXPECTED_ERR; /* get new sym */
                taux = AUX( symbol.itaux );     /* and itaux */
                if ( taux == NULL ) return SYM_UNEXPECTED_ERR;
            }
        if ( !( symbol.psymr->st == stStruct ||
                symbol.psymr->st == stUnion    ) )
                return SYM_UNEXPECTED_ERR;
      }
      return SYM_OK;
    }
    i++;            /* increment index to next symbol */
    if (( i >= lastSym ) && !symbol_found ) return SYM_VAR_NOT_FOUND;
  }
  return SYM_UNEXPECTED_ERR;
}
```

```
/*
 *              ===== countChars =====
 *
 * This function counts the number of times a particular character
 * (given by "Char") is found in the provided string "strg" in the
 * argument "cnt".
 */

static int countChars( char *strg, int *cnt, char Char )
{
/* counts the number of Char in the string */
  char *ptr;

  ptr = strg;
  *cnt = -1;
  do
    {
      ptr++;
      ptr = strchr( ptr, Char );
      (*cnt)++;
    }
  while( ptr != NULL );
  return SYM_OK;
}
```

```
/*
 *              ===== parseName =====
 *
 * This routine parses the provided variable name, and returns
 * indications of whether the name contains subelements
 * (expecting_struct) or indices (if num_indices > 0)
 */

int parseName(char **nextToken, char *myvarname,
              int *expecting_struct, int *numIndices, int **index_list )
{
  int   numchar;
  char  *lparenloc, *rparenloc, *lbrackloc, *rbrackloc, *seploc, *dotloc;
  char  *sepstrg = "[%d";
  char  sepchar = '[';
  enum   { none, C, Fortran } array_type;
  int   result;
  int   *indexPtr, indexCtr;

  *numIndices = 0;
  dotloc = strchr( myvarname, '.' );
  if (dotloc == NULL)
    {
      *expecting_struct = FALSE;
      *nextToken = myvarname;
    }
  else
    {
      *expecting_struct = TRUE;
      *dotloc = '\0';           /* separate token from rest of symbol */
      *nextToken = dotloc+1;
    }

  lbrackloc = strchr( myvarname, '[' ); /* look for C array */
  rbrackloc = strchr( myvarname, ']' );
  lparenloc = strchr( myvarname, '(' ); /*  "    " FORTRAN array */
  rparenloc = strchr( myvarname, ')' );

  if (   (lbrackloc == NULL)
      && (rbrackloc != NULL) ) return SYM_UNMATCHED_PAREN;
  if (   (lparenloc == NULL)
      && (rparenloc != NULL) ) return SYM_UNMATCHED_PAREN;

  array_type = none;

  if (lbrackloc != NULL)
    {
      if (rbrackloc == NULL) return SYM_UNMATCHED_PAREN;
      if (lparenloc != NULL) return SYM_BAD_SYNTAX;

      array_type = C;
    }

  if (lparenloc != NULL)
    {
      if (rparenloc == NULL) return SYM_UNMATCHED_PAREN;
      if (lbrackloc == NULL) return SYM_BAD_SYNTAX;

      lbrackloc = lparenloc;
      rbrackloc = rparenloc;

      sepstrg[0] = '(';

      array_type = Fortran;
    }

  if (array_type != none)
    {
      /* allocate memory for indexes */

      if (array_type == C)
        {
          result = countChars( lbrackloc, numIndices, ']' );
        }
      if (array_type == Fortran)
        {
          result = countChars( lbrackloc, numIndices, ',' );
          (*numIndices)++;
        }
      if (result != SYM_OK) return result;

      (*index_list) = malloc( (*numIndices)*sizeof( int ) );
      if (*index_list == NULL) return SYM_MEMORY_ERR;

      /* read first index */

      indexPtr = *index_list;
      indexCtr = 0;

      numchar = sscanf( lbrackloc, sepstrg, indexPtr );
      if (numchar < 1) return SYM_BAD_SYNTAX;
      if (array_type == Fortran) (*indexPtr)--;
      if (*indexPtr < 0) return SYM_BAD_SYNTAX;
      indexPtr++;
      indexCtr++;

      /* read remaining indexes */

      seploc = lbrackloc;
      if (array_type == Fortran)
        {
          sepchar = ',';
          sepstrg[0] = sepchar;
        }
      while (indexCtr < *numIndices)
        {
          seploc = strchr( seploc+1, sepchar );
          if (seploc == NULL) return SYM_BAD_SYNTAX;

          numchar = sscanf( seploc, sepstrg, indexPtr );
          if (numchar < 1) return SYM_BAD_SYNTAX;
          if (array_type == Fortran) (*indexPtr)--;
          if (*indexPtr < 0) return SYM_BAD_SYNTAX;

          indexPtr++;
          indexCtr++;
        }
      *lbrackloc = '\0';
    }

  return SYM_OK;
}
```

```c
/*
 *      ===== calcOffset =====
 *
 * This function is given the number of dimensions of
 * an array, as well as a list of the indexes in each dimension, and
 * returns the offset from the initial entry. It is limited to three
 * dimensions.  On entry, both "symbol" and "paux" have to be
 * initialized to point to the array entry and its associated
 * auxiliary symbol entry.  If any index is outside the allowable
 * dimensions, the routine returns SYM_INDEX_BOUNDS_ERR (something dbx
 * doesn't do).  If all goes well, the offset is stored in the
 * location pointed to by argument "offset", and returns SYM_OK. Any
 * other result should return SYM_UNEXPECTED_ERR.
 */

static int      calcOffset( long *offset, int num_indices, int *index_list )
{
  long  size, dimLo, dimHi;
  pAUXU dimpaux, paux;
  int j;


  *offset = 0;
  dimpaux = AUX( symbol.idaux );
  paux = dimpaux;
  if (paux[0].ti.bt == btTypedef) paux = paux+2; /* skip over extra RFD */
  switch( num_indices )
    {
        case 6: if (dimpaux->ti.tq5 != tqArray) return SYM_INDEX_BOUNDS_ERR;
        case 5: if (dimpaux->ti.tq4 != tqArray) return SYM_INDEX_BOUNDS_ERR;
        case 4: if (dimpaux->ti.tq3 != tqArray) return SYM_INDEX_BOUNDS_ERR;
        case 3: if (dimpaux->ti.tq2 != tqArray) return SYM_INDEX_BOUNDS_ERR;
        case 2: if (dimpaux->ti.tq1 != tqArray) return SYM_INDEX_BOUNDS_ERR;
        case 1: if (dimpaux->ti.tq0 != tqArray) return SYM_INDEX_BOUNDS_ERR;
            break;
        default: return SYM_UNEXPECTED_ERR;
    }
  for(j = num_indices-1; j>=0; j--)
  {
      dimLo = paux[3].dnLow;
      dimHi = paux[4].dnHigh;
      if (index_list[j] > dimHi) return SYM_INDEX_BOUNDS_ERR;
      if (index_list[j] < dimLo) return SYM_INDEX_BOUNDS_ERR;
      size = paux[5].width/8;
      *offset = (*offset) + index_list[j]*size;
      paux = paux+5; /* fall through to next dimension */
  }
  return SYM_OK;

}       /* end of calcOffset */
```

```c
/*
 *              ===== ls_findsym ====
 *
 * The main routine. Given a module name and variable name, this
 * routine looks up the address and type of variable and returns them
 * to the calling program. If a variable is global, "modname" must
 * consist of a single asterisk "*". The variable name can be a
 * scalar, array, or structure, with fields separated with periods
 * (customary C usage). Arrays can have no more than three
 * dimensions. This routine has been tested with C modules; no FORTRAN
 * support for structures or arrays is guaranteed. An appropriate
 * success or error code is returned (see ls_sym.h for the complete
 * list). If the initial attempt to access the debugger symbol table
 * fails, subsequent calls to ls_findsym will return the load error
 * message (either SYM_NO_SYMS or SYM_OPEN_ERR) will be returned
 * without further attempts to access the table.
 */

int ls_findsym( const char *modname, const char *varname,
                char **addr, vartype *vtype )
{
  static int    sym_load_status = SYM_NOT_LOADED;
  int           result, expecting_struct, elem_size;
  int           num_indices, *index_list = 0;
  long          offset;
  char          *tokenbuf, *myvarname, *tokenptr;
  char          *nextSym;
  size_t        stringsize;
  AUXU *taux; /*temporary*/

/* Module initialization */

  if (sym_load_status == SYM_NOT_LOADED)
      sym_load_status = loadSymbols();

  if (sym_load_status != SYM_OK) return sym_load_status;

  /* start search for symbol from beginning of file, or global section
     if modname is '*'      */

/* Lookup initialization */

  *addr = (char *) NULL;
  symbol_found = FALSE;
  module_found = FALSE;
  i = 0;
  symmax = symmaxlocal;
  if ( modname[0] == '*' )      /* global symbol requested */
    {
      module_found = TRUE;
      i = symmaxlocal;
      symmax = symmaxextern + i;
      end_of_proc = symmax - 1;
    }

/* Ready to do lookup */

  if ( !module_found )
    {
      result = lookForModule( modname );
      if (result != SYM_OK) return result;
    }

/* make local copy of variable name */
```

```
    stringsize = strlen( varname );
    tokenbuf = malloc( stringsize+1 );
    if ( tokenbuf == NULL ) return SYM_MEMORY_ERR;
    myvarname = tokenbuf;
    strncpy( myvarname, varname, stringsize );
    myvarname[stringsize] = '\0';   /* make sure the string is terminated */
    tokenptr = myvarname;         /* initialize parser pointer */

    expecting_struct = FALSE;

  /* loop until symbol found and not expecting a structure */

    while(!symbol_found || expecting_struct)
        /* same as !(sym_fnd && !exp_strct) */
      {
        /* parse name into tokens */

        result = parseName( &nextSym, tokenptr, &expecting_struct,
                            &num_indices, &index_list );
        if (result != SYM_OK) SYMEXIT( result );

        /* look for next required symbol */

        result = lookForSym( tokenptr, symbol_found,
                             expecting_struct, addr, num_indices );
        if (result != SYM_OK) SYMEXIT( result );

        switch( AUX( symbol.itaux )->ti.bt )
            {
            case btChar:   *vtype = Char;  elem_size = sizeof( char               ); break
;
            case btUChar:  *vtype = UChar; elem_size = sizeof( unsigned char      ); break
;
            case btShort:  *vtype = SHint; elem_size = sizeof( short int          ); break
;
            case btUShort: *vtype = USHint;elem_size = sizeof( unsigned short int ); break
;
            case btInt:    *vtype = Sint;  elem_size = sizeof( int                ); break
;
            case btUInt:   *vtype = Uint;  elem_size = sizeof( int                ); break
;
            case btLong:   *vtype = Slng;  elem_size = sizeof( long               ); break
;
            case btULong:  *vtype = Ulng;  elem_size = sizeof( long               ); break
;
            case btFloat:  *vtype = flt;   elem_size = sizeof( float              ); break
;
            case btDouble: *vtype = dbl;   elem_size = sizeof( double             ); break
;
            default:
                {
                  *vtype = Unknown;
                  if (!expecting_struct) SYMEXIT( SYM_NOT_SCALAR );
                  elem_size = symbol.psymr->value/8;
                }
            }

        /* calculate address of indexed element, if any */

        if (num_indices > 0)
            {
                result = calcOffset( &offset, num_indices, index_list );
                if (result != SYM_OK) SYMEXIT( result );
                *addr = (*addr) + offset;
            }

        tokenptr = nextSym;
      }

    free( tokenbuf );
    free( index_list );

    if (*vtype == Unknown) return SYM_UNEXPECTED_ERR;

    return SYM_OK;
}


void ls_print_findsym_error(int result, char *mod_name, char *var_name)
/* Prints an appropriate error on stderr if result is non-zero */
{
    fprintf(stderr, "Error in routine ls_findsym: ");
    switch ( result )
        {
        case SYM_UNEXPECTED_ERR:
            fprintf(stderr,
                    "Unexpected error encountered when\n\tlooking up variable '%s' i
n module '%s'.\n",
                    var_name, mod_name);
            fprintf(stderr,
                    "\tPossible indexing of scalar variable?\n");
            break;
        case SYM_OPEN_ERR:
            fprintf(stderr,
                    "Error opening symbol table.\n");
            break;
        case SYM_NO_SYMS:
            fprintf(stderr,
                    "Symbol table not found.\n");
            break;
        case SYM_MOD_NOT_FOUND:
            fprintf(stderr,
                    "Module '%s' not found.\n", mod_name);
            break;
        case SYM_VAR_NOT_FOUND:
            fprintf(stderr,
                    "Variable '%s' not found in module '%s'.\n", var_name, mod_name)
;
            break;
        case SYM_NOT_SCALAR:
            fprintf(stderr,
                    "Variable '%s' in module '%s' is non-scalar. Facility variables
must be scalar.\n",
                    var_name, mod_name );
            break;
        case SYM_NOT_STATIC:
            fprintf(stderr,
                    "Variable '%s' in module '%s' must be declared static to be used
 in facilities.\n",
                    var_name, mod_name );
            break;
        case SYM_MEMORY_ERR:
            fprintf(stderr,
                    "Memory error in ls_findsym routine; couldn't allocate something
.\n");
            break;
        case SYM_UNMATCHED_PAREN:
            fprintf(stderr,
                    "Unmatched parenthesis found when\n\tlooking for variable '%s' i
```

```
n module '%s'.\n",
                    var_name, mod_name);
                break;
            case SYM_BAD_SYNTAX:
                fprintf(stderr,
                    "Bad syntax when looking for variable '%s' in module '%s'.\n",
                    var_name, mod_name);
                break;
            case SYM_INDEX_BOUNDS_ERR:
                fprintf(stderr,
                    "Symbol indexing bounds error detected when\n\tlooking for '%s' in m
odule '%s'.\n",
                    var_name, mod_name);
                break;
            default:
                fprintf(stderr,
                    "Unrecognized error code %d returned while looking for '%s\%s'.\n",
                    result, var_name, mod_name);
        } /* end of switch (result) statement */
}


double  ls_get_double(vartype sym_type, void *addr )
    /* obtains data at addr and returns a double value, based on type given in sym_type
*/
{
    double value = 1./0.;       /* Generate Inf */

    if (addr)
        {
            switch( sym_type )
                {
                case Char:  value = *(  signed       char   *) addr; break;
                case UChar: value = *(unsigned       char   *) addr; break;
                case SHint: value = *(  signed short int    *) addr; break;
                case USHint:value = *(unsigned short int    *) addr; break;
                case Sint:  value = *(  signed       int    *) addr; break;
                case Uint:  value = *(unsigned       int    *) addr; break;
                case Slng:  value = *(  signed long  int    *) addr; break;
                case Ulng:  value = *(unsigned long         *) addr; break;
                case  flt:  value = *(               float  *) addr; break;
                case  dbl:  value = *(               double *) addr; break;
                } /* end of switch( sym_type ) statement */
        }
    return value;
}


void ls_set_double(vartype sym_type, void *addr, double value )
    /* Sets variable at addr to value of double provided */
{
    switch( sym_type )
        {
        case Char:  *(  signed       char   *) addr = value; break;
        case UChar: *(unsigned       char   *) addr = value; break;
        case SHint: *(  signed short int    *) addr = value; break;
        case USHint:*(unsigned short int    *) addr = value; break;
        case Sint:  *(  signed       int    *) addr = value; break;
        case Uint:  *(unsigned       int    *) addr = value; break;
        case Slng:  *(  signed long  int    *) addr = value; break;
        case Ulng:  *(unsigned long         *) addr = value; break;
        case  flt:  *(               float  *) addr = value; break;
        case  dbl:  *(               double *) addr = value; break;
        } /* end of switch( sym_type ) statement */
}


double  ls_get_sym_val( symbol_rec *symrec, int *error )

        /* This routine attempts to return the present value of the symbol
           described in symbol_rec. If Addr is non-zero, the value of that
           location, interpreted as type double, will be returned. If Addr
           is zero, and Mod_Name and Par_Name are both not null strings,
           the ls_findsym() routine is used to try to obtain the address
           by looking at debugger symbol tables in the executable image, and
           the value of the double contained at that address is returned,
           and the symbol record is updated to contain the address of that
           symbol. If an error is discovered, 'error' will be non-zero and
           and error message is printed on stderr.                       */
{
    *error = 0;

    if (!symrec->Addr)
        {
            /* Here on null address; look up symbol in tables */

            *error = ls_findsym( symrec->Mod_Name, symrec->Par_Name,
                                 &symrec->Addr, &symrec->Par_Type );

            if ((!*error) && (!symrec->Addr))   /* still null addr */
                *error = SYM_UNEXPECTED_ERR;

            if (*error) /* report any problems and give up */
                {
                    ls_print_findsym_error( *error,
                                            symrec->Mod_Name,
                                            symrec->Par_Name );
                    return *error;
                }
        }

    /* here on non-NULL address */

    return ls_get_double(symrec->Par_Type, symrec->Addr);

}


void    ls_set_sym_val( symbol_rec *symrec, double value )

        /* This routine sets the value of a double at the location pointed
           to by the symbol_rec's Addr field, if Addr is non-zero. If Addr
           is zero, and Mod_Name and Par_Name are both not null strings,
           the ls_findsym() routine is used to try to obtain the address
           by looking at debugger symbol tables in the executable image, and
           the value of the double contained at that address is returned,
           and the symbol record is updated to contain the address of that
           symbol. If an error is discovered, 'error' will be non-zero and
           and error message is printed on stderr.                      */
{
    int error;

    if (!symrec->Addr)
        {
            /* Here on null address; look symbol in tables */

            error = ls_findsym( symrec->Mod_Name, symrec->Par_Name,
                                &symrec->Addr, &symrec->Par_Type );
```

```
        if ((!error) && (!symrec->Addr))    /* still null addr */
                    error = SYM_UNEXPECTED_ERR;
        if (error)
            {
                ls_print_findsym_error( error,
                                        symrec->Mod_Name,
                                        symrec->Par_Name );
                return;
            }
    }

    /* here on non-NULL address */

    ls_set_double( symrec->Par_Type, symrec->Addr, value );

    return;
}
```

```
/***************************************************************************

        TITLE:  ls_sync.c

----------------------------------------------------------------------------

        FUNCTION:       Real-time synchronization routines for LaRCSIM

----------------------------------------------------------------------------

        MODULE STATUS:  Developmental

----------------------------------------------------------------------------

        GENEALOGY:      Written 921229 by Bruce Jackson

----------------------------------------------------------------------------

        DESIGNED BY:    EBJ

        CODED BY:       EBJ

        MAINTAINED BY:  EBJ

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        930104  Added ls_resync() call to avoid having to pass DT as a
                global variable.                                EBJ
        940204  Added calculation of sim_control variable overrun to
                indicate a frame overrun has occurred.          EBJ
        940506  Added support for sim_control_.debug flag, which disables
                synchronization (there is still a local dbg flag that
                enables synch error logging)                    EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_sync.c,v 1.7 1994/05/06 15:34:54 bjax Stab $
$Log: ls_sync.c,v $
 * Revision 1.7  1994/05/06  15:34:54  bjax
 * Removed "freerun" variable, and substituted sim_control_.debug flag.
 *
 * Revision 1.6  1994/02/16  13:01:22  bjax
 * Added logic to signal frame overrun; corrected syntax on ls_catch call
 * (old style may be BSD format). EBJ
 *
 * Revision 1.5  1993/07/30  18:33:14  bjax
 * Added 'dt' parameter to call to ls_sync from ls_resync routine.
 *
 * Revision 1.4  1993/03/15  14:56:13  bjax
 * Removed call to ls_pause; this should be done in cockpit routine.
 *
 * Revision 1.3  93/03/13  20:34:09  bjax
 * Modified to allow for sync times longer than a second; added ls_pause()  EBJ
 *
 * Revision 1.2  93/01/06  09:50:47  bjax
 * Added ls_resync() function.
 *
 * Revision 1.1  92/12/30  13:19:51  bjax
 * Initial revision
 *
 * Revision 1.3  93/12/31  10:34:11  bjax
 * Added $Log marker as well.
 *

----------------------------------------------------------------------------

        REFERENCES:

----------------------------------------------------------------------------

        CALLED BY:

----------------------------------------------------------------------------

        CALLS TO:

----------------------------------------------------------------------------

        INPUTS:

----------------------------------------------------------------------------

        OUTPUTS:

---------------------------------------------------------------------------*/
#include <sys/time.h>
#include <signal.h>
#include <stdio.h>
#include "ls_types.h"
#include "ls_sim_control.h"


extern SCALAR Simtime;

/* give the  time interval data structure FILE visibility */

static struct itimerval t, ot;

static int dbug = 0;

/*void ls_catch( sig, code, sc) /* signal handler */
/*int sig;
int code;
struct sigcontext *sc;*/
void ls_catch()
{
  static DATA lastSimtime = -99.9;

  /*if (lastSimtime == Simtime) fprintf(stderr, "Overrun.\n"); */
  if (dbug) printf("ls_catch called\n");
  sim_control_.overrun = (lastSimtime == Simtime);
  lastSimtime = Simtime;
  signal(SIGALRM, ls_catch);
}

void ls_sync(dt)
float dt;

/* this routine syncs up the interval timer for a new dt value */
{
  int terr;
  int isec;
  float usec;

  if (sim_control_.debug!=0) return;
```

```
  isec = (int) dt;
  usec = 1000000* (dt - (float) isec);

  t.it_interval.tv_sec = isec;
  t.it_interval.tv_usec = usec;
  t.it_value.tv_sec = isec;
  t.it_value.tv_usec = usec;
  if (dbug) printf("ls_sync called\n");
  ls_catch();   /* set up for SIGALRM signal catch */
  terr = setitimer( ITIMER_REAL, &t, &ot );
  if (terr) perror("Error returned from setitimer");
}

void ls_unsync()
/* this routine unsyncs the interval timer */
{
  int terr;

  if (sim_control_.debug!=0) return;
  t.it_interval.tv_sec = 0;
  t.it_interval.tv_usec = 0;
  t.it_value.tv_sec = 0;
  t.it_value.tv_usec = 0;
if (dbug) printf("ls_unsync called\n");

  terr = setitimer( ITIMER_REAL, &t, &ot );
  if (terr) perror("Error returned from setitimer");

}

void ls_resync()
/* this routine resynchronizes the interval timer to the old
   interrupt period, stored in struct ot by a previous call
   to ls_unsync().  */
{
  float dt;

  if (sim_control_.debug!=0) return;
  if (dbug) printf("ls_resync called\n");
  dt = ((float) ot.it_interval.tv_usec)/1000000. +
       ((float) ot.it_interval.tv_sec);
  ls_sync(dt);
}

void ls_pause()
/* this routine waits for the next interrupt */
{
  if (sim_control_.debug!=0) return;
  if (dbug) printf("ls_pause called\n");
  pause();
}
```

```
/***************************************************************************

        TITLE:          ls_trim.c

----------------------------------------------------------------------------

        FUNCTION:       Trims the simulated aircraft by using certain
                        controls to null out a similar number of outputs.

        This routine used modified Newton-Raphson method to find the vector
        of control corrections, delta_U, to drive a similar-sized vector of
        output errors, Y, to near-zero.  Nearness to zero is to within a
        tolerance specified by the Criteria vector.  An optional Weight
        vector can be used to improve the numerical properties of the
        Jacobian matrix (called H_Partials).

        Using a single-sided difference method, each control is
        independently perturbed and the change in each output of
        interest is calculated, forming a Jacobian matrix H (variable
        name is H_Partials):

                        dY = H dU


        The columns of H correspond to the control effect; the rows of
        H correspond to the outputs affected.

        We wish to find dU such that for U = U0 + dU,

                        Y = Y0 + dY = 0
                        or dY = -Y0

        One solution is dU = inv(H)*(-Y0); however, inverting H
        directly is not numerically sound, since it may be singular
        (especially if one of the controls is on a limit, or the
        problem is poorly posed).  An alternative is to either weight
        the elements of dU to make them more normalized; another is to
        multiply both sides by the transpose of H and invert the
        resulting [H' H].  This routine does both:

                        -Y0  =      H dU
                     W (-Y0) =    W H dU        premultiply by W
                  H' W (-Y0) = H' W H dU        premultiply by H'

              dU = [inv(H' W H)][ H' W (-Y0)]   Solve for dU

        As a further refinement, dU is limited to a smallish magnitude
        so that Y approaches 0 in small steps (to avoid an overshoot
        if the problem is inherently non-linear).

        Lastly, this routine can be easily fooled by "local minima",
        or depressions in the solution space that don't lead to a Y =
        0 solution.  The only advice we can offer is to "go somewheres
        else and try again"; often approaching a trim solution from a
        different (non-trimmed) starting point will prove beneficial.


----------------------------------------------------------------------------

        MODULE STATUS:  developmental

----------------------------------------------------------------------------

        GENEALOGY:      Created from old CASTLE SHELL$TRIM.PAS
                        on 6 FEB 95, which was based upon an Ames
                        CASPRE routine called BQUIET.

----------------------------------------------------------------------------

        DESIGNED BY:    E. B. Jackson

        CODED BY:       same

        MAINTAINED BY:  same

----------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                                 BY

        950307  Modified to make use of ls_get_sym_val and ls_put_sym_val
                routines.                                               EBJ
        950329  Fixed bug in making use of more than 3 controls;
                removed call by ls_trim_get_set() to ls_trim_init(). EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_trim.c,v 1.9 1995/03/29 16:09:56 bjax Exp $
$Log: ls_trim.c,v $
 * Revision 1.9  1995/03/29  16:09:56  bjax
 * Fixed bug in having more than three trim controls; removed unnecessary
 * call to ls_trim_init in ls_trim_get_set. EBJ
 *
 * Revision 1.8  1995/03/16  12:28:40  bjax
 * Fixed problem where ls_trim() returns non-zero if
 * symbols are not loaded - implies vehicle trimmed when
 * actually no trim attempt is made. This results in storing of non-
 * trimmed initial conditions in sims without defined trim controls.
 *
 * Revision 1.7  1995/03/15  12:17:12  bjax
 * Added flag marker line to ls_trim_put_set() routine output.
 *
 * Revision 1.6  1995/03/08  11:49:07  bjax
 * Minor improvements to ls_trim_get_set; deleted weighting parameter
 * for output definition; added comment lines to settings file output.
 *
 * Revision 1.5  1995/03/07  22:38:04  bjax
 * Removed ls_generic.h; this version relies entirely on symbol table routines to
 * set and get variable values. Added additional fields to Control record structure;
 * created Output record with appropriate fields. Added ls_trim_put_set() and
 * ls_trim_get_set() routines. Heavily modified initialization routine; most of this
 * logic now resides in ls_trim_get_set(). Renamed all routines so that they being
 * with "ls_trim_" to avoid conflicts.
 *  EBJ
 *
 * Revision 1.4  1995/03/07  13:04:16  bjax
 * Configured to use ls_get_sym_val() and ls_set_sym_val().
 *
 * Revision 1.3  1995/03/03  01:59:53  bjax
 * Moved definition of SYMBOL_NAME and SYMBOL_TYPE to ls_sym.h
 * and removed from this module. EBJ
 *
 * Revision 1.2  1995/02/27  19:53:41  bjax
 * Moved symbol routines to ls_sym.c to declutter this file. EBJ
 *
 * Revision 1.1  1995/02/27  18:14:10  bjax
 * Initial revision
 *
```

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

        REFERENCES:

---------------------------------------------------------------------------

        CALLED BY:

---------------------------------------------------------------------------

        CALLS TO:

---------------------------------------------------------------------------

        INPUTS:

---------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

static char rcsid[] = "$Id: ls_trim.c,v 1.9 1995/03/29 16:09:56 bjax Exp $";


#include <string.h>
#include "ls_constants.h"
#include "ls_types.h"
#include "ls_sym.h"
#include "ls_matrix.h"


#ifndef TRUE
#define FALSE  0
#define TRUE  !FALSE
#endif

#define MAX_NUMBER_OF_CONTROLS 10
#define MAX_NUMBER_OF_OUTPUTS 10
#define STEP_LIMIT 0.01
#define NIL_POINTER 0L

#define FACILITY_NAME_STRING "trim"
#define CURRENT_VERSION 10


typedef struct
{
    symbol_rec  Symbol;
    double      Min_Val, Max_Val, Curr_Val, Authority;
    double      Percent, Requested_Percent, Pert_Size;
    int         Ineffective, At_Limit;
} control_rec;

typedef struct
{
    symbol_rec  Symbol;
    double      Curr_Val, Weighting, Trim_Criteria;
    int         Uncontrollable;
} output_rec;


static  int             Symbols_loaded = 0;
static  int             Index;
static  int             Trim_Cycles;
static  int             First;
static  int             Trimmed;
static  double          Gain;

static  int             Number_of_Controls;
static  int             Number_of_Outputs;
static  control_rec     Controls[ MAX_NUMBER_OF_CONTROLS ];
static  output_rec      Outputs[ MAX_NUMBER_OF_OUTPUTS ];

static  double          **H_Partials;

static  double          Baseline_Output[ MAX_NUMBER_OF_OUTPUTS ];
static  double          Saved_Control, Saved_Control_Percent;

static  double          Cost, Previous_Cost;



int ls_trim_init()
/*  Initialize partials matrix */
{
    int i, error;
    int result;

    Index = -1;
    Trim_Cycles = 0;
    Gain = 1;
    First = 1;
    Previous_Cost = 0.0;
    Trimmed = 0;

    for (i=0;i<Number_of_Controls;i++)
        {
            Controls[i].Curr_Val = ls_get_sym_val( &Controls[i].Symbol, &error );
            if (error) Controls[i].Symbol.Addr = NIL_POINTER;
            Controls[i].Requested_Percent =
                (Controls[i].Curr_Val - Controls[i].Min_Val)
                /Controls[i].Authority;
        }

    H_Partials = nr_matrix( 1, Number_of_Controls, 1, Number_of_Controls );
    if (H_Partials == 0) return -1;

    return 0;
}

void ls_trim_get_vals()
/* Load the Output vector, and calculate control percent used */
{
    int i, error;

    for (i=0;i<Number_of_Outputs;i++)
        {
            Outputs[i].Curr_Val = ls_get_sym_val( &Outputs[i].Symbol, &error );
            if (error) Outputs[i].Symbol.Addr = NIL_POINTER;
        }

    Cost = 0.0;
    for (i=0;i<Number_of_Controls;i++)
        {
            Controls[i].Curr_Val = ls_get_sym_val( &Controls[i].Symbol, &error );
            if (error) Controls[i].Symbol.Addr = NIL_POINTER;
            Controls[i].Percent =
                (Controls[i].Curr_Val - Controls[i].Min_Val)
```

```
                /Controls[i].Authority;
        }


}

void ls_trim_move_controls()
/* This routine moves the current control to specified percent of authority */
{
    int i;

    for(i=0;i<Number_of_Controls;i++)
        {
            Controls[i].At_Limit = 0;
            if (Controls[i].Requested_Percent <= 0.0)
                {
                    Controls[i].Requested_Percent = 0.0;
                    Controls[i].At_Limit = 1;
                }
            if (Controls[i].Requested_Percent >= 1.0)
                {
                    Controls[i].Requested_Percent = 1.0;
                    Controls[i].At_Limit = 1;
                }
            Controls[i].Curr_Val = Controls[i].Min_Val +
                (Controls[i].Max_Val - Controls[i].Min_Val) *
                Controls[i].Requested_Percent;
        }
}

void ls_trim_put_controls()
/* Put current control requests out to controls themselves */
{
    int i;

    for (i=0;i<Number_of_Controls;i++)
            if (Controls[i].Symbol.Addr)
                ls_set_sym_val( &Controls[i].Symbol, Controls[i].Curr_Val );
}

void ls_trim_calc_cost()
/* This routine calculates the current distance, or cost, from trim */
{
    int i;

    Cost = 0.0;
    for(i=0;i<Number_of_Outputs;i++)
        Cost += pow((Outputs[i].Curr_Val/Outputs[i].Trim_Criteria),2.0);
}

void ls_trim_save_baseline_outputs()
{
    int i, error;

    for (i=0;i<Number_of_Outputs;i++)
            Baseline_Output[i] = ls_get_sym_val( &Outputs[i].Symbol, &error );
}

int  ls_trim_eval_outputs()
{
    int i, trimmed;

    trimmed = 1;
    for(i=0;i<Number_of_Outputs;i++)
        if( fabs(Outputs[i].Curr_Val) > Outputs[i].Trim_Criteria) trimmed = 0;
    return trimmed;
}

void ls_trim_calc_h_column()
{
    int i;
    double delta_control, delta_output;

    delta_control = (Controls[Index].Curr_Val - Saved_Control)/Controls[Index].Autho
rity;

    for(i=0;i<Number_of_Outputs;i++)
        {
            delta_output = Outputs[i].Curr_Val - Baseline_Output[i];
            H_Partials[i+1][Index+1] = delta_output/delta_control;
        }
}

void ls_trim_do_step()
{
    int i, j, l, singular;
    double      **h_trans_w_h;
    double      delta_req_mag, scaling;
    double      delta_U_requested[ MAX_NUMBER_OF_CONTROLS ];
    double      temp[ MAX_NUMBER_OF_CONTROLS ];

    /* Identify ineffective controls (whose partials are all near zero) */

    for (j=0;j<Number_of_Controls;j++)
        {
            Controls[j].Ineffective = 1;
            for(i=0;i<Number_of_Outputs;i++)
                if (fabs(H_Partials[i+1][j+1]) > EPS) Controls[j].Ineffective = 0;
        }

    /* Identify uncontrollable outputs */

    for (j=0;j<Number_of_Outputs;j++)
        {
            Outputs[j].Uncontrollable = 1;
            for(i=0;i<Number_of_Controls;i++)
                if (fabs(H_Partials[j+1][i+1]) > EPS) Outputs[j].Uncontrollable = 0;
        }

    /* Calculate well-conditioned partials matrix [ H' W H ] */

    h_trans_w_h = nr_matrix(1, Number_of_Controls, 1, Number_of_Controls);
    if (h_trans_w_h == 0)
        {
            fprintf(stderr, "Memory error in ls_trim().\n");
            exit(1);
        }
    for (l=1;l<=Number_of_Controls;l++)
        for (j=1;j<=Number_of_Controls;j++)
            {
                h_trans_w_h[l][j] = 0.0;
                for (i=1;i<=Number_of_Outputs;i++)
                    h_trans_w_h[l][j] +=
                        H_Partials[i][l]*H_Partials[i][j]*Outputs[i-1].Weighting;
            }

    /* Invert the partials array  [ inv( H' W H ) ]; note: h_trans_w_h is replaced
       with its inverse during this function call */

    singular = nr_gaussj( h_trans_w_h, Number_of_Controls, 0, 0 );
```

```
    if (singular) /* Can't invert successfully */
        {
            nr_free_matrix( h_trans_w_h, 1, Number_of_Controls,
                                         1, Number_of_Controls );
            fprintf(stderr, "Singular matrix in ls_trim().\n");
            return;
        }

    /* Form right hand side of equality: temp = [ H' W (-Y) ] */

    for(i=0;i<Number_of_Controls;i++)
        {
            temp[i] = 0.0;
            for(j=0;j<Number_of_Outputs;j++)
                temp[i] -= H_Partials[j+1][i+1]*Baseline_Output[j]*Outputs[j].Weighting;
        }

    /* Solve for dU = [inv( H' W H )][ H' W (-Y)] */
    for(i=0;i<Number_of_Controls;i++)
        {
            delta_U_requested[i] = 0.0;
            for(j=0;j<Number_of_Controls;j++)
                delta_U_requested[i] += h_trans_w_h[i+1][j+1]*temp[j];
        }

    /* limit step magnitude to certain size, but not direction */

    delta_req_mag = 0.0;
    for(i=0;i<Number_of_Controls;i++)
        delta_req_mag += delta_U_requested[i]*delta_U_requested[i];
    delta_req_mag = sqrt(delta_req_mag);
    scaling = STEP_LIMIT/delta_req_mag;
    if (scaling < 1.0)
        for(i=0;i<Number_of_Controls;i++)
            delta_U_requested[i] *= scaling;

    /* Convert deltas to percent of authority */

    for(i=0;i<Number_of_Controls;i++)
        Controls[i].Requested_Percent = Controls[i].Percent + delta_U_requested[i];

    /* free up temporary matrix */

    nr_free_matrix( h_trans_w_h, 1, Number_of_Controls,
                                 1, Number_of_Controls );

}


int ls_trim()
{
    const int Max_Cycles = 100;
    int Baseline;

    Trimmed = 0;
    if (Symbols_loaded) {

        ls_trim_init();                /* Initialize Outputs & controls */
        ls_trim_get_vals();  /* Limit the current control settings */
        Baseline = TRUE;
        ls_trim_move_controls();                /* Write out the new values of controls
*/
        ls_trim_put_controls();
        ls_loop( 0.0, -1 );            /* Cycle the simulation once with new limite
d
                                          controls */

        /* Main trim cycle loop follows */

        while((!Trimmed) && (Trim_Cycles < Max_Cycles))
            {
                ls_trim_get_vals();
                if (Index == -1)
                    {
                        ls_trim_calc_cost();
                        /*Adjust_Gain();        */
                        ls_trim_save_baseline_outputs();
                        Trimmed = ls_trim_eval_outputs();
                    }
                else
                    {
                        ls_trim_calc_h_column();
                        Controls[Index].Curr_Val = Saved_Control;
                        Controls[Index].Percent  = Saved_Control_Percent;
                        Controls[Index].Requested_Percent = Saved_Control_Percent;
                    }
                Index++;
                if (!Trimmed)
                    {
                        if (Index >= Number_of_Controls)
                            {
                                Baseline = TRUE;
                                Index = -1;
                                ls_trim_do_step();
                            }
                        else
                            { /* Save the current value & pert next control */
                                Baseline = FALSE;
                                Saved_Control = Controls[Index].Curr_Val;
                                Saved_Control_Percent = Controls[Index].Percent;

                                if (Controls[Index].Percent <
                                    (1.0 - Controls[Index].Pert_Size) )
                                    {
                                        Controls[Index].Requested_Percent =
                                            Controls[Index].Percent +
                                            Controls[Index].Pert_Size ;
                                    }
                                else
                                    {
                                        Controls[Index].Requested_Percent =
                                            Controls[Index].Percent -
                                            Controls[Index].Pert_Size;
                                    }
                            }
                        ls_trim_move_controls();
                        ls_trim_put_controls();
                        ls_loop( 0.0, -1 );
                        Trim_Cycles++;
                    }
            }

        nr_free_matrix( H_Partials, 1, Number_of_Controls, 1, Number_of_Controls );
    }

    if (!Trimmed)  fprintf(stderr, "Trim unsuccessful.\n");
    return Trimmed;
```

```
}

char *ls_trim_get_set(char *buffer, char *eob)
/* This routine parses the settings file for "trim" entries. */
{

    static char *fac_name = FACILITY_NAME_STRING;
    char *bufptr, **lasts, *nullptr, null = '\0';
    char line[256];
    int n, ver, i, error, abrt;
    enum {controls_header, controls, outputs_header, outputs, done} looking_for;

    nullptr = &null;
    lasts = &nullptr;
    abrt = 0;
    looking_for = controls_header;


    n = sscanf(buffer, "%s", line);
    if (n == 0) return 0L;
    if (strncasecmp( fac_name, line, strlen(fac_name) )) return 0L;

    bufptr = strtok_r( buffer+strlen(fac_name)+1, "\n", lasts);
    if (bufptr == 0) return 0L;

    sscanf( bufptr, "%d", &ver );
    if (ver != CURRENT_VERSION) return 0L;

    while( !abrt && (eob > bufptr))
      {
        bufptr = strtok_r( 0L, "\n", lasts );
        if (bufptr == 0) return 0L;
        if (strncasecmp( bufptr, "end", 3) == 0) break;

        sscanf( bufptr, "%s", line );
        if (line[0] != '#') /* ignore comments */
            {
                switch (looking_for)
                    {
                    case controls_header:
                        {
                            if (strncasecmp( line, "controls", 8) == 0)
                                {
                                    n = sscanf( bufptr, "%s%d", line, &Number_of_Control
s );

                                    if (n != 2) abrt = 1;
                                    looking_for = controls;
                                    i = 0;
                                }
                            break;
                        }
                    case controls:
                        {
                            n = sscanf( bufptr, "%s%s%le%le%le",
                                        Controls[i].Symbol.Mod_Name,
                                        Controls[i].Symbol.Par_Name,
                                        &Controls[i].Min_Val,
                                        &Controls[i].Max_Val,
                                        &Controls[i].Pert_Size);
                            if (n != 5) abrt = 1;
                            Controls[i].Symbol.Addr = NIL_POINTER;
                            i++;
                            if (i >= Number_of_Controls) looking_for = outputs_header;
                            break;
                        }
                    case outputs_header:
                        {
                            if (strncasecmp( line, "outputs", 7) == 0)
                                {
                                    n = sscanf( bufptr, "%s%d", line, &Number_of_Out
puts );

                                    if (n != 2) abrt = 1;
                                    looking_for = outputs;
                                    i = 0;
                                }
                            break;
                        }
                    case outputs:
                        {
                            n = sscanf( bufptr, "%s%s%le",
                                        Outputs[i].Symbol.Mod_Name,
                                        Outputs[i].Symbol.Par_Name,
                                        &Outputs[i].Trim_Criteria );
                            if (n != 3) abrt = 1;
                            Outputs[i].Symbol.Addr = NIL_POINTER;
                            i++;
                            if (i >= Number_of_Outputs) looking_for = done;
                        }
                    case done:
                        {
                            break;
                        }
                    }

            }
      }

    if ((!abrt) &&
        (Number_of_Controls > 0) &&
        (Number_of_Outputs == Number_of_Controls))
        {
            Symbols_loaded = 1;

            for(i=0;i<Number_of_Controls;i++) /* Initialize fields in Controls data
*/
                {
                    Controls[i].Curr_Val = ls_get_sym_val( &Controls[i].Symbol, &err
or );

                    if (error)
                        Controls[i].Symbol.Addr = NIL_POINTER;
                    Controls[i].Authority = Controls[i].Max_Val - Controls[i].Min_Va
l;

                    if (Controls[i].Authority == 0.0)
                        Controls[i].Authority = 1.0;
                    Controls[i].Requested_Percent =
                        (Controls[i].Curr_Val - Controls[i].Min_Val)
                        /Controls[i].Authority;
                    Controls[i].Pert_Size = Controls[i].Pert_Size/Controls[i].Author
ity;
                }

            for (i=0;i<Number_of_Outputs;i++) /* Initialize fields in Outputs data *
/
                {
                    Outputs[i].Curr_Val = ls_get_sym_val( &Outputs[i].Symbol, &error
 );

                    if (error) Outputs[i].Symbol.Addr = NIL_POINTER;
                    Outputs[i].Weighting =
                        Outputs[0].Trim_Criteria/Outputs[i].Trim_Criteria;
```

```
            }
        }

    bufptr = *lasts;
    return bufptr;
}


void ls_trim_put_set( FILE *fp )
{
    int i;

    if (fp==0) return;
    fprintf(fp, "\n");
    fprintf(fp, "#==============================  %s\n", FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, FACILITY_NAME_STRING);
    fprintf(fp, "\n");
    fprintf(fp, "%04d\n", CURRENT_VERSION);
    fprintf(fp, "  controls: %d\n", Number_of_Controls);
    fprintf(fp, "#    module    parameter   min_val   max_val   pert_size\n");
    for (i=0; i<Number_of_Controls; i++)
        fprintf(fp, "    %s\t%s\t%E\t%E\t%E\n",
                Controls[i].Symbol.Mod_Name,
                Controls[i].Symbol.Par_Name,
                Controls[i].Min_Val,
                Controls[i].Max_Val,
                Controls[i].Pert_Size*Controls[i].Authority);
    fprintf(fp, "  outputs: %d\n", Number_of_Outputs);
    fprintf(fp, "#    module    parameter   trim_criteria\n");
    for (i=0;i<Number_of_Outputs;i++)
        fprintf(fp, "    %s\t%s\t%E\n",
                Outputs[i].Symbol.Mod_Name,
                Outputs[i].Symbol.Par_Name,
                Outputs[i].Trim_Criteria );
    fprintf(fp, "end\n");
    return;
}
```

```
/*************************************************************************

        TITLE:          ls_writeasc1.c

--------------------------------------------------------------------------

        FUNCTION:       Writes out time history data GetData asc1 format.

--------------------------------------------------------------------------

        MODULE STATUS:  developmental

--------------------------------------------------------------------------

        GENEALOGY:      940510 Bruce Jackson

        WRITTEN BY:     EBJ

        CODED BY:       EBJ

        MAINTAINED BY:  EBJ

--------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        940510  Generated from ls_writemat.c                    EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_writeasc1.c,v 1.7 1995/04/07 01:44:34 bjax Exp $
$Log: ls_writeasc1.c,v $
 * Revision 1.7  1995/04/07  01:44:34  bjax
 * Added logic to avoid endless loop if wrapped and Tape->Last == Tape->Length.
 *
 * Revision 1.6  1995/03/03  01:55:53  bjax
 * Modified to use new def'n of Tape->Chan structure (includes symbol rec
 * defined in ls_sym.h). EBJ
 *
 * Revision 1.5  1994/05/11  13:27:01  bjax
 * Added check to avoid writing bad channels.
 *
 * Revision 1.4  1994/05/11  12:39:53  bjax
 * Changed format of time stamp to get better resolution.
 *
 * Revision 1.3  1994/05/11  11:57:35  bjax
 * Shortened variable names to 15 chars to avoid error in xp import;
 * also eliminated structure names (just use field names).
 *
 * Revision 1.2  1994/05/10  20:58:44  bjax
 * Forced variable names to be 16 chars or shorter.
 *
 * Revision 1.1  1994/05/10  20:10:39  bjax
 * Initial revision
 *

--------------------------------------------------------------------------

        REFERENCES:

--------------------------------------------------------------------------

        CALLED BY:

--------------------------------------------------------------------------

        CALLS TO:

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

--------------------------------------------------------------------------*/

#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include "ls_types.h"
#include "ls_tape.h"
#include "ls_sim_control.h"


/*      ============
         definitions
        ============    */
#define MAX_STRING 255
#define NIL_POINTER 0L
#define NAMES_PER_LINE 5
#define PTS_PER_LINE 4

extern TAPE     *Tape;
extern char *progname;

char *GDname( const char *inname, char *outname )

{
    /* delete non-alphanumeric characters */

    char * buf;

    buf = outname;
    while ( *inname )
    {
        if   ( ( (*inname >= 'A') && (*inname <= 'Z' ) )
            || ( (*inname >= 'a') && (*inname <= 'z' ) )
            || ( (*inname >= '0') && (*inname <= '9' ) )
            || ( *inname == '_' ))
                *outname++ = *inname;
        if (*inname == '.') buf = outname; /* ignore structure names */
        inname++;
    }
    *outname = '\0';
    buf[15] = '\0';
    return buf;

}

                            /*---------------+
===========================>|  ls_writeasc1  |<============================
                            +---------------*/

void ls_writeasc1( char *file_name)

{
```

```
    int     wrapped = 1;
    long    names_written, pts_written;
    int     i, j, null_chans;
    FILE    *fp;
    char    namebuf[128];

/* Count the number of null channels */

    null_chans = 0;
    for (i = 0; i<Tape->Num_Chan; i++)
        if (Tape->Chan[i]->Symbol.Addr == NULL) null_chans++;

    if (Tape->Num_Chan - null_chans > 0)
      {

        fp = fopen(file_name, "w");

/* Write header info */

        fprintf(fp, "format  asc 1   .1      \n");
        fprintf(fp, "title   %s\n", progname);
        fprintf(fp, "nChans  %d\n", Tape->Num_Chan - null_chans);

/* Write data names
 *  and emulate FORTRAN format statement:
 *
 *     format(a8,8x,4a16:/(5a16))
 *
 */


        fprintf(fp, "names           ");
        names_written = 1;
        for (j = 0; j<Tape->Num_Chan; j++)
          if ( Tape->Chan[j]->Symbol.Addr != NULL)
            {
                fprintf(fp, "%-16s",
                        GDname( Tape->Chan[j]->Symbol.Par_Name,
                                namebuf ));
                names_written++;
                if (names_written >= NAMES_PER_LINE)
                    {
                        names_written = 0;
                        fprintf(fp, "\n");
                    }
            }
        if (names_written != 0) fprintf(fp, "\n");

/* Write data values */

        fprintf(fp, "data001 \n");

/* Need to emulate fortran format statement:
 *
 *  format(f10.3,10x,3g20.14:/(4g20.14))
 *
 */

        i = Tape->First;
        if (Tape->First < Tape->Last) wrapped = 0;

        while( wrapped || (i <= Tape->Last) )
            {
                pts_written = 1;
                fprintf( fp, "% 10.3E          ", Tape->T_Stamp[i] );
                for (j = 0; j<Tape->Num_Chan; j++)
                    if ( Tape->Chan[j]->Symbol.Addr != NULL)
                        {
                            fprintf(fp, "% 20.14E", Tape->Chan[j]->Data[i] );
                            pts_written++;
                            if (pts_written >= PTS_PER_LINE)
                                {
                                    pts_written = 0;
                                    fprintf(fp, "\n");
                                }

                        }
                if (pts_written != 0) fprintf(fp, "\n");
                i = i + sim_control_.write_spacing;
                if ( i >= Tape->Length )
                    if ( wrapped )
                        {
                            i = i - Tape->Length;
                            wrapped = 0;
                        }
                    else
                        {
                            i = Tape->Last+1;
                        }
            }

/* Write trailer info */

        fclose( fp );

      } /* end of Num_Chans > 0 */
}
```

```
/*************************************************************************

        TITLE:  ls_writeav.c

--------------------------------------------------------------------------

        FUNCTION:       Writes out time history data in Agile-VU flight format

--------------------------------------------------------------------------

        MODULE STATUS:  Developmental

--------------------------------------------------------------------------

        GENEALOGY:      Written 921230 by Bruce Jackson (see Mod History)

--------------------------------------------------------------------------

        DESIGNED BY:    EBJ

        CODED BY:       EBJ

        MAINTAINED BY:  EBJ

--------------------------------------------------------------------------

        MODIFICATION HISTORY:

        DATE    PURPOSE                                         BY

        920507  Converted from Trajectory Viewer                EBJ
        920806  Converted from ASC2_to_PO16; ported to IRIS     EBJ
        920810  Incorporated single file argument               EBJ
        921027  Modified and incorporated into LaRCSim code     EBJ
        930915  Modified to correct runway start time for long runs     EBJ
        931220  Cleaned up the time slice array access          EBJ
        940111  Fixed include files; was ls_eom.h; also changed
                DATA to SCALAR types.                           EBJ
        940509  Changed so wrapped tapes are handled properly; also
                obey the output interpolation flag "write_spacing".     EBJ

        CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_writeav.c,v 1.10 1995/04/07 01:44:34 bjax Exp $
$Log: ls_writeav.c,v $
 * Revision 1.10  1995/04/07  01:44:34  bjax
 * Added logic to avoid endless loop if wrapped and Tape->Last == Tape->Length.
 *
 * Revision 1.9  1994/05/11  19:47:11  bjax
 * Added support for multiple runs; unfortunately, the runway has to
 * be drawn last, which means it covers up the aircraft on top of it.
 * Talk about priority problems!
 *
 * Revision 1.8  1994/05/09  21:20:26  bjax
 * Added support for wrapped tape buffer and output interpolation.
 *
 * Revision 1.7  1994/01/11  19:03:12  bjax
 * Fixed include files; changed DATA to SCALAR type.
 *
 * Revision 1.6  1993/12/20  16:48:32  bjax
 * Cleaned up the time slice array access method.  EBJ
 *
 * Revision 1.4  1993/09/15  20:42:37  bjax
 * Modified code to put correct start time on 'runway' vehicle time stamp.
 *
 * Revision 1.3  1993/08/03  19:59:28  bjax
 * Modified to use current state variables being stored in Tape.  EBJ
 *
 * Revision 1.2  1993/06/08  10:31:35  bjax
 * Changed runway heading from -30 to 0 deg.  EBJ
 *
 * Revision 1.1  92/12/30  13:20:35  bjax
 * Initial revision
 *
 * Revision 1.3  93/12/31  10:34:11  bjax
 * Added $Log marker as well.
 *

--------------------------------------------------------------------------

        REFERENCES:

--------------------------------------------------------------------------

        CALLED BY:

--------------------------------------------------------------------------

        CALLS TO:

--------------------------------------------------------------------------

        INPUTS:

--------------------------------------------------------------------------

        OUTPUTS:

-------------------------------------------------------------------------*/


/*      ================
         include files
        ================        */

#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include "ls_types.h"
#include "ls_tape.h"
#include "ls_sim_control.h"


/*      =============
         definitions
        =============   */
#define MAX_STRING 255
#define NIL_POINTER 0L

/* temporary defs */
#define IALT 15
#define IPSI 8
#define IPHI 6
#define ITHETA 7
#define ISX 13
#define ISY 14


extern TAPE *Tape;
extern char *progname;
```

```
av_write_header( FILE *fp )
  {
        static int init = 0;
        static int header_num = 0;

        if (init == 0 )  /* should only have one source line */
            {
                init = -1;
                fprintf(fp, "Source:    '%s ACES simulation'\n", progname);
            }

        header_num++;

/* Write header info */

        fprintf(fp, "Type:      %s\n", progname);
        fprintf(fp, "Role:      Blue\n");
        fprintf(fp, "Name:      Run%03d\n", header_num);
        fprintf(fp, "Parameters:\n");
        fprintf(fp, "XCG        'X Position'      ft          0 -50000.00   50000.00
25000.00\n");
        fprintf(fp, "YCG        'Y Position'      ft          0 -50000.00   50000.00
25000.00\n");
        fprintf(fp, "ZCG        'Z Position'      ft          0      0.00  -30000.00
10000.00\n");
        fprintf(fp, "PHIB       'Roll Angle'      deg         1   -180.00     180.00
    45.00\n");
        fprintf(fp, "THETAB     'Pitch Angle'     deg         1    -90.00      90.00
    45.00\n");
        fprintf(fp, "PSIB       'Heading Angle'   deg         1   -180.00     180.00
    45.00\n");
  }


av_write_rwy_object( FILE *fp, SCALAR t_min, SCALAR t_max )
  {
        fprintf(fp, "Type:      Rwy\n");
        fprintf(fp, "Role:      Blue\n");
        fprintf(fp, "Name:      Runway\n");
        fprintf(fp, "Parameters:\n");
        fprintf(fp, "XCG        'X Position'      ft          0 -50000.00   50000.00
25000.00\n");
        fprintf(fp, "YCG        'Y Position'      ft          0 -50000.00   50000.00
25000.00\n");
        fprintf(fp, "ZCG        'Z Position'      ft          0      0.00  -30000.00
10000.00\n");
        fprintf(fp, "PHIB       'Roll Angle'      deg         1   -180.00     180.00
    45.00\n");
        fprintf(fp, "THETAB     'Pitch Angle'     deg         1    -90.00      90.00
    45.00\n");
        fprintf(fp, "PSIB       'Heading Angle'   deg         1   -180.00     180.00
    45.00\n");
        fprintf(fp, "Time:      %f\n", t_min);
        fprintf(fp, "0  0       0       0.000000        0         0.000000\n");
        fprintf(fp, "Time:      %f\n", t_max);
        fprintf(fp, "0  0       0       0.000000        0         0.000000\n");
  }

                                /*---------------+
==========================>|   ls_writeav    |<==============================
                                +---------------*/

void ls_writeav( char *file_name)
{
        int     wrapped = 1;
        int     i, j;
        FILE    *fp;
        SCALAR  xic, yic, zic;
        SCALAR  t_min, t_max, t_last;

/* Protect against using bad Tape */

        if (Tape == NULL) return;

        fp = fopen(file_name, "w");

        av_write_header( fp );

        i = Tape->First;

        if (Tape->First < Tape->Last) wrapped = 0;

        t_min = Tape->T_Stamp[i];
        t_max = Tape->T_Stamp[i];
        t_last = Tape->T_Stamp[i];

        while( wrapped || (i <= Tape->Last) )
        {
            if (Tape->T_Stamp[i] < t_last) av_write_header( fp );
            if (Tape->T_Stamp[i] < t_min ) t_min = Tape->T_Stamp[i];
            if (Tape->T_Stamp[i] > t_max ) t_max = Tape->T_Stamp[i];
            t_last = Tape->T_Stamp[i];
            fprintf(fp, "Time:\t%f\n", Tape->T_Stamp[i] );
            fprintf(fp, "%f\t%f\t%f\t",
                                Tape->Chan[ ISX     ]->Data[ i ],
                                Tape->Chan[ ISY     ]->Data[ i ],
                               -Tape->Chan[ IALT    ]->Data[ i ]);
            fprintf(fp, "%f\t%f\t%f\n",
                        57.3*Tape->Chan[ IPHI    ]->Data[ i ],
                        57.3*Tape->Chan[ ITHETA  ]->Data[ i ],
                        57.3*Tape->Chan[ IPSI    ]->Data[ i ]);

            i = i + sim_control_.write_spacing;
            if ( i >= Tape->Length )
                    if ( wrapped )
                        {
                            i = i - Tape->Length;
                            wrapped = 0;
                        }
                    else
                        {
                            i = Tape->Last+1;
                        }
        }

        av_write_rwy_object( fp, t_min, t_max );

        fclose( fp );
}
```

```
/***************************************************************************

	TITLE:	ls_writemat.c

----------------------------------------------------------------------------

	FUNCTION:	Writes out time history data in ASCII matrix form.
			Can thereafter be read by several popular matrix
			laboratory software packages.

----------------------------------------------------------------------------

	MODULE STATUS:	developmental

----------------------------------------------------------------------------

	GENEALOGY:	921230 Bruce Jackson (see mod history below)

	WRITTEN BY:	EBJ

	CODED BY:	EBJ

	MAINTAINED BY:	EBJ

----------------------------------------------------------------------------

	MODIFICATION HISTORY:

	DATE	PURPOSE						BY

	920507	Converted from Trajectory Viewer		EBJ
	920806	Converted from ASC2_to_PO16; ported to IRIS	EBJ
	920810	Incorporated single file argument		EBJ
	921027	Modified and incorporated into LaRCSim code	EBJ
	921029	Copied from  ls_writeav.c and modded to write matrix.	EBJ
	930803	Modified to work with the Tape parameter names	EBJ
	931220	Cleaned up the timeslice vector access method	EBJ
	940111	Changed header files from ls_eom.h to ls_types.	EBJ
	940506	Corrected handling of wrapped tapes (I think).	EBJ
	940509	More fixes to wrapped tapes.. maybe this works..	EBJ
	940511	Added handling of null data channels and better
		mapping of variable names into matlab names	EBJ

	CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_writemat.c,v 1.11 1995/04/07 01:44:34 bjax Exp $
$Log: ls_writemat.c,v $
 * Revision 1.11  1995/04/07  01:44:34  bjax
 * Added logic to avoid endless loop if wrapped and Tape->Last == Tape->Length.
 *
 * Revision 1.10  1995/03/03  01:58:42  bjax
 * Modified to use new def'n of Tape->Chan structure (includes symbol rec
 * defined in ls_sym.h) EBJ
 *
 * Revision 1.9  1994/05/11  13:50:04  bjax
 * Added correction to skip over null channels. Also map C
 * variable names into matlab names without structure prefixes.
 *
 * Revision 1.8  1994/05/10  11:56:20  bjax
 * Allow underscore in variable names.
 *
 * Revision 1.7  1994/05/09  21:19:49  bjax
 * Added support for wrapped tape buffers and output interpolation.
 *
 * Revision 1.6  1994/05/06  18:25:24  bjax
 * Corrected wrapping of tape (I think).
 *
 * Revision 1.5  1994/01/11  19:04:25  bjax
 * Changed header file from "
 * "ls_eom.h" to "ls_types.h".
 *
 * Revision 1.4  1993/12/20  16:50:21  bjax
 * Cleaned up the time slice access method.  EBJ
 *
 * Revision 1.3  1993/12/04  12:39:52  bjax
 * minor fixes.
 *
 * Revision 1.2  1993/08/03  19:18:02  bjax
 * Writemat now uses tape header information to write out matrix file.  EBJ
 *
 * Revision 1.1  1992/12/30  13:20:51  bjax
 * Initial revision
 *
 * Revision 1.3  93/12/31  10:34:11  bjax
 * Added $Log marker as well.
 *

----------------------------------------------------------------------------

	REFERENCES:

----------------------------------------------------------------------------

	CALLED BY:

----------------------------------------------------------------------------

	CALLS TO:

----------------------------------------------------------------------------

	INPUTS:

----------------------------------------------------------------------------

	OUTPUTS:

--------------------------------------------------------------------------*/

#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include "ls_types.h"
#include "ls_tape.h"
#include "ls_sim_control.h"


/*	============
	 definitions
	============	*/
#define MAX_STRING 255
#define NIL_POINTER 0L


extern TAPE	*Tape;

char *matname( const char *inname, char *outname )

{
```

```
    /* delete non-alphanumeric characters */

    char * buf;

    buf = outname;
    while ( *inname )
    {
        if    ( ( (*inname >= 'A') && (*inname <= 'Z' ) )
              || ( (*inname >= 'a') && (*inname <= 'z' ) )
              || ( (*inname >= '0') && (*inname <= '9' ) )
              || ( *inname == '_' ))
                *outname++ = *inname;
        if (*inname == '.') buf = outname; /* ignore structure names */
        inname++;
    }
    *outname = '\0';
    buf[15] = '\0';
    return buf;

}

                            /*---------------+
===========================>|  ls_writemat   |<===============================
                            +---------------*/

void ls_writemat( char *file_name)

{
        int     wrapped = 1;
        int     i, j, null_chans;
        FILE    *fp;
        char    namebuf[128];

/* Count the number of null channels */

    null_chans = 0;
    for (i = 0; i<Tape->Num_Chan; i++)
        if (Tape->Chan[i]->Symbol.Addr == NULL) null_chans++;

    if ( Tape->Num_Chan - null_chans > 0)
      {
        fp = fopen(file_name, "w");

        i = Tape->First;

        if (Tape->First < Tape->Last) wrapped = 0;

/* Write header info */

        fprintf(fp, "temp = [\n");

        while( wrapped || (i <= Tape->Last) )
        {

                fprintf( fp, "%f", Tape->T_Stamp[i] );

                for ( j = 0; j < Tape->Num_Chan; j++ )
                  if (Tape->Chan[j]->Symbol.Addr != NULL)
                    fprintf( fp, "\t%f", Tape->Chan[j]->Data[i] );

                fprintf( fp, "\n" );

                i = i + sim_control_.write_spacing;
                if ( i >= Tape->Length )
                    if ( wrapped )
                        {
                            i = i - Tape->Length;
                            wrapped = 0;
                        }
                    else
                        {
                            i = Tape->Last+1;
                        }
        }

/* Write trailer info */

        fprintf( fp, "];\nt     = temp(:, 1);\n" );
        i = 2;
        for( j = 0; j < Tape->Num_Chan; j++ )
          if (Tape->Chan[j]->Symbol.Addr != NULL)
            fprintf( fp, "%s = temp(:, %d);\n",
                     matname(Tape->Chan[j]->Symbol.Par_Name, namebuf), i++ );
        fprintf(fp, "clear temp\n");
        fclose( fp );
      }
}
```

```
/***************************************************************************

	TITLE:		ls_writetab.c

----------------------------------------------------------------------------

	FUNCTION:	Writes out time history data in ASCII tab-delimited
			format, which can thereafter be read by several
			popular graphing packages.

----------------------------------------------------------------------------

	MODULE STATUS:	developmental

----------------------------------------------------------------------------

	GENEALOGY:	940510 E. B. Jackson

	WRITTEN BY:	EBJ

	CODED BY:	EBJ

	MAINTAINED BY:	EBJ

----------------------------------------------------------------------------

	MODIFICATION HISTORY:

	DATE	PURPOSE						BY

	940510	Copied from ls_writemat.c			EBJ

	CURRENT RCS HEADER:

$Header: /aces/larcsim/dev/RCS/ls_writetab.c,v 1.4 1995/04/07 01:44:34 bjax Exp $
$Log: ls_writetab.c,v $
 * Revision 1.4  1995/04/07  01:44:34  bjax
 * Added logic to avoid endless loop if wrapped and Tape->Last == Tape->Length.
 *
 * Revision 1.3  1995/03/03  01:57:51  bjax
 * Modified to use new def'n of Tape->Chan structure (includes symbol rec
 * defined in ls_sym.h)  EBJ
 *
 * Revision 1.2  1994/05/11  13:50:58  bjax
 * Added fix to skip over null channels.
 *
 * Revision 1.1  1994/05/10  11:56:39  bjax
 * Initial revision
 *

----------------------------------------------------------------------------

	REFERENCES:

----------------------------------------------------------------------------

	CALLED BY:

----------------------------------------------------------------------------

	CALLS TO:
		matname - found in ls_writemat.c - maps C names to ascii

----------------------------------------------------------------------------

	INPUTS:

----------------------------------------------------------------------------

	OUTPUTS:

--------------------------------------------------------------------------*/

#include <getopt.h>
#include <stdio.h>
#include <string.h>
#include "ls_types.h"
#include "ls_tape.h"
#include "ls_sim_control.h"


/*	============
	 definitions
	============	*/
#define MAX_STRING 255
#define NIL_POINTER 0L


extern TAPE	*Tape;

extern char *matname( const char *inname, char *outname );
/* defined in ls_writemat.c */


                             /*---------------+
===========================>|  ls_writetab    |<==============================
                             +---------------*/

void ls_writetab( char *file_name)

{
	int	wrapped = 1;
	int	i, j, null_chans;
	FILE	*fp;
	char	namebuf[128];


/* Count the number of null channels */

    null_chans = 0;
    for (i = 0; i<Tape->Num_Chan; i++)
	if (Tape->Chan[i]->Symbol.Addr == NULL) null_chans++;

    if (Tape->Num_Chan - null_chans > 0)
      {
	fp = fopen(file_name, "w");

	i = Tape->First;

	if (Tape->First < Tape->Last) wrapped = 0;

/* Write header info */

	fprintf( fp, "Time" );
	for ( j = 0; j < Tape->Num_Chan; j++ )
	  if (Tape->Chan[j]->Symbol.Addr != NULL)
	    fprintf( fp, "\t%s", matname(Tape->Chan[j]->Symbol.Par_Name, namebuf) );
	fprintf( fp, "\n" );

/* Write data */
```

```
        while( wrapped || (i <= Tape->Last) )
        {

                fprintf( fp, "%f", Tape->T_Stamp[i] );

                for ( j = 0; j < Tape->Num_Chan; j++ )
                  if (Tape->Chan[j]->Symbol.Addr != NULL)
                    fprintf( fp, "\t%f", Tape->Chan[j]->Data[i] );

                fprintf( fp, "\n" );

                i = i + sim_control_.write_spacing;
                if ( i >= Tape->Length )
                    if ( wrapped )
                        {
                            i = i - Tape->Length;
                            wrapped = 0;
                        }
                    else
                        {
                            i = Tape->Last+1;
                        }
        }

        fclose( fp );
      }
}
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1995	Technical Memorandum

4. TITLE AND SUBTITLE
Manual for a Workstation-based Generic Flight Simulation Program (LaRCsim) Version 1.4

5. FUNDING NUMBERS
505-64-52-01

6. AUTHOR(S)
E. Bruce Jackson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA TM - 110164

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified-Unlimited

Subject Category - 08

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

LaRCsim is a set of ANSI C routines that implement a full set of equations of motion for a rigid-body aircraft in atmospheric and low-earth orbital flight, suitable for pilot-in-the-loop simulations on a workstation-class computer. All six rigid-body degrees of freedom are modeled. The modules provided include calculations of the typical aircraft rigid body simulation variables, earth geodesy, gravity and atmosphere models, and support several data recording options. Features/limitations of the current version include English units of measure, a 1962 atmosphere model in cubic spline function lookup form, ranging from sea level to 75,000 feet, rotating oblate spheroidal earth model, with aircraft C.G. coordinates in both geocentric and geodetic axes. Angular integrations are done using quaternion angular state variables. Vehicle X-Z symmetry is assumed.

14. SUBJECT TERMS
flight simulation, UNIX, real-time, equations of motion, source code, oblate spheroid, and flight dynamics

15. NUMBER OF PAGES
141

16. PRICE CODE
A07

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102