SAE TECHNICAL PAPER SERIES

940923

An Overview of the HVE Developer's Toolkit

Terry D. Day
Engineering Dynamics Corp.

Reprinted from: Accident Reconstruction: Technology and Animation IV (SP-1030)

International Congress & Exposition
Detroit, Michigan
February 28-March 3, 1994

400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A. Tel: (412)776-4841 Fax:(412)776-5760

ISSN 0148-7191

Printed in USA 90-1203C/PG

940923

An Overview of the HVE Developer's Toolkit

Terry D. Day
Engineering Dynamics Corp.

ABSTRACT

A substantial programming effort is required to develop a human or vehicle dynamics simulator. More than half of this effort is spent designing and programming the user interface (the means by which the user supplies program input and views program output). This paper describes a pre-programmed, 3-dimensional (3-D), input/output window-type interface which may be used by developers of human and vehicle dynamics programs. By using this interface, the task of input/output programming is reduced by approximately 50 percent, while simultaneously providing a more robust interface. This paper provides a conceptual overview of the interface, as well as specific details for writing human and vehicle dynamics programs which are compatible with the interface. Structures are provided for the human, vehicle and environment models. Structures are also provided for events, interface variables, and the output data stream. By using these standardized structures, any compatible physics model (i.e., human or vehicle dynamics simulator) may be linked into the window interface to model and illustrate, using fully-rendered, 3-D scientific visualization, the kinematic and kinetic behavior of humans and vehicles within their environment.

SIMULATION IS USEFUL for studying the response of humans and vehicles during a crash. In addition, simulation is useful for studying vehicular response to driver inputs and environmental factors before and after a crash, as well as in non-crash environments. For these reasons, safety researchers have devoted significant resources toward the development of simulation tools, both for human simulation [1, 2, 3]* and vehicle simulation [4 - 13]. Each of these simulation tools was originally developed under contract with U.S. government agencies (FHWA, NHTSA) or the Motor Vehicle Manufacturers Association.

* Numbers in brackets designate references found at the end of the paper.

Simulation programs consist of several individual components. These components may be broadly categorized into two areas: *Physics* and *Interface*.

The physics components include a controlling routine, a numerical integration routine, a force/moment calculation routine and a derivative calculation routine. See reference 14 for the design details of a typical simulation program. The interface components include an input routine, an output routine and, possibly, a graphics routine. Printing and plotting routines may also be required, if not supplied by the computer's operating system.

Researchers wishing to develop a human or vehicle dynamics program must necessarily include in their program all the interface components; this task is not trivial. A review of several existing simulations (see Table 1) reveals that the number of lines of interface code typically represents approximately 51 percent of the total programming for a batch-oriented program using numeric input/output. For a menu-driven program with numeric input/output and 2-D graphics, the percentage of interface code grows to 67 percent (see Table 2). Thus, only one third to one half of the code produced by the researchers is actually related to the physics of simulation.

This paper describes a 3-dimensional software interface toolkit for use by developers of human and vehicle simulation programs. The purpose of this toolkit is to reduce the development time and effort while significantly improving the quality and usefulness of the simulation. A review of simulation programs using this interface (see Table 3) reveals the amount of input/output code in the simulation is reduced to 24 percent. This reduction is possible because the programmer is able to leverage off of several hundreds of thousands of lines of pre-programmed interface code developed specifically for use by human and vehicle dynamics simulation programs.

This paper provides an overview of this interface, called HVE (*Human-Vehicle-Environment*), and describes how to write human and vehicle dynamics programs which are compatible with the interface.

Table 1. Percentage of Input/Output Code for Several Existing Mainframe Simulation and Reconstruction Programs

Program	Type	2/3D	Interface Method	Graphics	Percent of Total Program Physics	I/O
CRASH3	2-vehicle collision reconstruction	2-D	Menu	None	44	56
SMAC	2-vehicle collision simulation	2-D	Batch	Wireframe	49	51
TBST	1-vehicle simulation	2-D	Menu	None	40	60
TBSTT	articulated vehicle simulation	2-D	Menu	None	40	60
PHASE4	articulated vehicle simulation	3-D	Batch	None	43	57
HSRI-3D	occupant/pedestrian simulation	3-D	Batch	None	59	41
HVOSM	1-vehicle simulation	3-D	Batch	None	67	33
				Average	49	51

Table 2. Percentage of Input/Output Code for Several Existing PC Simulation and Reconstruction Programs

Program	Type	2/3D	Interface Method	Graphics	Percent of Total Program * Physics	I/O
EDCRASH	2-vehicle collision reconstruction	2-D	Menu	Wireframe	32	68
EDSMAC	2-vehicle collision simulation	2-D	Menu	Wireframe	46	54
EDSVS	1-vehicle simulation	2-D	Menu	Wireframe	26	74
EDVTS	articulated vehicle simulation	2-D	Menu	Wireframe	27	73
				Average	33	67

* These percentages do not include 103 kilobytes of I/O code in EDVAP shared libraries

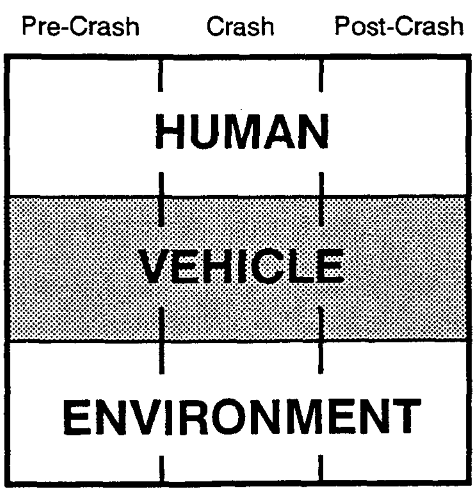


Figure 1 - Nine-cell Matrix For Accident Reconstruction

Overview of HVE

HVE is a computer abstraction of the nine-cell matrix for accident reconstruction (see figure 1) originally proposed by the late Dr. William Haddon, first Director of the National Highway Traffic Safety Administration [15]. The nine-cell matrix describes the possible interactions between humans, vehicles and their environment during the pre-crash, crash and post-crash phases of an accident.

HVE is not itself an accident reconstruction program. Rather, HVE is an interface for running accident reconstruction and simulation programs, much like Microsoft Windows™ is an interface for running PC programs.

The HVE interface is an integrated set of editors for creating 3-D physical and visual models of humans, vehicles and environments. Once created, the kinematics and kinetics of these models may be analyzed by any

Table 3. Percentage of Input/Output Code for Programs in Table 1 When Using HVE Toolkit

Program	Type	2D/3D	Interface Method	Graphics	Percent of Total Program* Physics	I/O
EDCRASH	2-vehicle collision reconstruction	†	Window	‡	82	18
EDSMAC	2-vehicle collision simulation	†	Window	‡	85	15
EDSVS	1-vehicle simulation	†	Window	‡	70	30
EDVTS	articulated vehicle simulation	†	Window	‡	71	29
EDVDS	articulated vehicle simulation	3-D	Window	‡	64	36
EDHIS	occupant/pedestrian simulation	3-D	Window	‡	76	24
EDVSM	1-vehicle simulation	3-D	Window	‡	83	17
				Average	76	24

* Current estimates; programming is not yet complete

† All conventional, 2-D methods use the 3-D HVE physical road surfaces, which may be sloped. Thus, these methods become quasi-3-D. See references 16 and 17 for additional information.

‡ All graphic images are in color (up to 16.7 M colors), rendered (lighted and phong shaded), and may be viewed from any user-specified position.

HVE-compatible human or vehicle dynamics program. The program results may be displayed both numerically and *visually* by HVE. Sequences of several events produced from several runs may be edited into a single coherent accident sequence using the HVE playback editor. The output may be routed to a display, printer, plotter or VCR. See reference 16 for further details.

The remainder of this document describes the details of producing HVE-compatible simulation programs.

Toolkit Description

A block diagram for a typical HVE-compatible physics model (human or vehicle dynamics calculation method) is shown in figure 2. Note that conceptually, the HVE interface *surrounds* the physics model.

The HVE Developer's Toolkit is a library of functions and data structures that provide the developer of a human

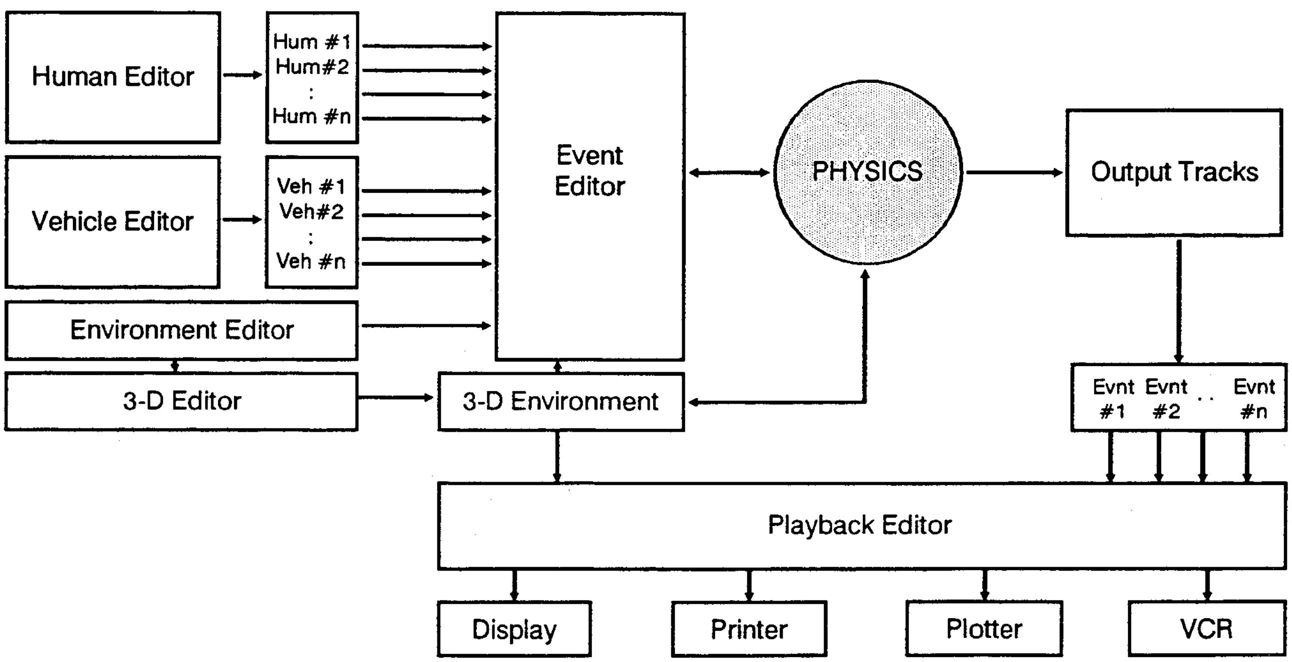


Figure 2 - Block Diagram for HVE application environment. Note that some of the arrows are bi-directional.

Table 4. HVE Toolkit Library Functions

Function Name	*Arguments*	*Description*	*Called By*	*Calls*
HveMain()	*(none)*	*main routine*	*HVE Interface*	*InitMessageQueue()* *send()* *receive()* *ReceiveHveInput()* *ParseHveInput()* *CalcMethodInfo()* *OutputTrackSetup()* *ExecuteCalcMethod()* *ShutDown()*
ReceiveHveInput()	*(none)*	*reads all HVE data into data structures*	*HveMain()*	*send()* *receive()*
SendHveOutput()	*number of humans, number of vehicles*	*sends all simulation results for current time-step back to HVE*	*Physics*	*send()* *receive()*
GetSurfaceInfo()	$(X,Y,Z)_{tire}$, $\Psi_{tire\ plane}$, *friction factor,* $(i,j,k)_{tire}$	*based on current* $(X,Y)_{tire}$*, returns* Z_{tire}*, friction and slope from the HVE 3-D physical environment to the physics*	*physics*	*send()* *receive()*
InitMessageQueue()	*(none)*	*establishes communication between HVE Interface and physics*	*HveMain()*	*(n/a)*
send()	*pointer to data, number of bytes of data*	*message to tell HVE Interface that data is coming*	*HveMain()* *ReceiveHveInput()* *SendHveOutput()*	*(n/a)* *GetSurfaceInfo()*
receive()	*pointer to data, number of bytes of data*	*transfer data from physics to HVE Interface*	*HveMain()* *ReceiveHveInput()*	*(n/a)* *SendHveOutput()* *GetSurfaceInfo()*

or vehicle dynamics program access to the HVE interface. These functions and data structures are described below.

HVE Functions

The functions included in the HVE Developer's Toolkit are shown in Table 4. Using these functions makes the calculation method HVE-compatible. Such methods may then use the HVE 3-dimensional human, vehicle and environment models, as well as the event and playback editors. All HVE input devices (mouse, scanner) and output devices (display, printer, plotter, VCR/VTR) are also available to the method.

All HVE-compatible methods will include one HveMain function. HveMain is called by the HVE interface when the physics program is selected to ensure the user's input data are compatible with the programmer's physics program. HveMain is also called when the user executes the event. HveMain performs the task of assigning all the HVE human, vehicle and environment data to the physics program (using the ReceiveHveInput function).

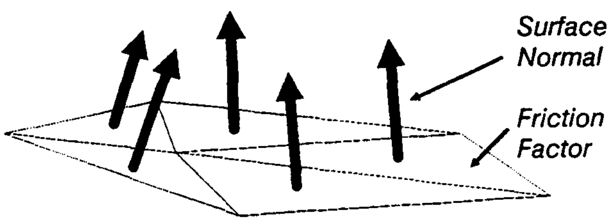
The programmer must provide five functions which are called by HveMain. These are: ParseHveInput, OutputTrackSetup, CalcMethodInfo, ExecuteCalcMethod and ShutDown. (See Table 5 for a brief description of these functions.) However, all other tasks performed by HveMain, such as setting up the message queue and sending data back and forth between the HVE interface and the physics, are transparent to the programmer.

Table 5. User-supplied Functions Required by the HVE Toolkit

Function Name	*Arguments*	*Description*	*Called By*	*Calls*
ParseHveInput()	*(none)*	*converts HVE interface data into values used by physics*	*HveMain()*	*(none required)*
OutputTrackSetup()	*(none)*	*selects individual output variables the physics will send back to HVE, and if it will be user-editable*	*HveMain()*	*(none)*
CalcMethodInfo()	*(none)*	*selects which HVE event dialogs (positions,driver tables, etc.) and output types trajectory sim., variable output, vehicle data, etc.) are made available to the user*	*HveMain()*	*(none)*
ExecuteCalcMethod()	*(none)*	*initializes and executes the calculation method*	*HveMain()*	*(physics-dependent)*
ShutDown()	*(none)*	*send results summary and warning messages to the HVE interface for output during playback,*	*HveMain()*	*(none)*

The HVE Developer's Toolkit requires the programmer to supply the human or vehicle dynamics calculation method (reconstruction or simulation model). The calculation method must include the SendHveOutput function in its output routine. This function sends the physics' output data structure back to the HVE interface for playback.

The road surface in the HVE environment is composed of a series of polygons, created graphically during the process of creating the 3-D environment. As shown in figure 3, each of these polygons has slope, elevation and friction properties.



Each Environment Surface Polygon Has:
- Elevation, Z = f(X,Y)
- Surface Normal Vector
- Friction Factor

Figure 3 - Tire-Environment Interaction Model

Although not required, the calculation method may also include the GetSurfaceInfo function into its physics model. This function is used to obtain from the 3-D environment the current elevation, slope and friction information for the specified earth-fixed X,Y coordinates (see figure 3). By incorporating the GetSurfaceInfo function, the calculation method becomes much more robust, because it can use 3-D information to provide elevation, roll and pitch data to 2-D *and* 3-D methods. It also provides a virtually unlimited number of terrain boundaries.

For further details, see *Procedure For Developing HVE-Compatible Applications*, found later in this paper.

HVE Data Structures

The data structures included in the HVE Developer's Toolkit are shown in Table 6. These structures are the means by which all data are passed between the HVE interface and the calculation method. They are used to:

- send/receive HVE input data
- send/receive HVE output data
- send/receive road surface data

Table 6. HVE Toolkit Data Structures (see also Appendix A for details of each data structure)

Structure Name	*Description*	*Used By (function name)*
HumanData	*contains all data describing the HVE human model*	*ReceiveHveInput()* *ParseHveInput()*
VehicleData	*contains all data describing the HVE vehicle model*	*ReceiveHveInput()* *ParseHveInput()*
EnvironmentData	*contains all data describing the HVE environment model*	*ReceiveHveInput()* *ParseHveInput()*
EventData	*contains all data describing the HVE event (calculation options, selected humans and/or vehicles, restraint system options, and output options)*	*ReceiveHveInput()* *ParseHveInput()* *CalcMethodInfo()*
EventHumanData	*contains all event-related data describing each HVE human in the event*	*ReceiveHveInput()* *ParseHveInput()*
EventVehicleData	*contains all event-related data describing each HVE vehicle in the event*	*ReceiveHveInput()* *ParseHveInput()*
InterfaceData	*contains all current simulation options (integration data, output and playback time intervals)*	*ReceiveHveInput()* *ParseHveInput()*
HumanOutputTrackSetup	*contains setup information regarding each potential human output variable (number of variables and usage)*	*OutputTrackSetup()*
VehicleOutputTrackSetup	*contains setup information regarding each potential vehicle output variable (number of variables and usage)*	*OutputTrackSetup()*
OutputHumanData	*contains human output track data for current timestep*	*SendHveOutput()*
OutputVehicleData	*contains vehicle output track data for current timestep*	*SendHveOutput()*
SurfaceInfoData	*contains surface geometrical and frictional data for tire force calculations*	*GetSurfaceInfo()*

OutputHumanData and OutputVehicleData communicate between the physics output routine and the HVE interface at each timestep. SurfaceInfoData communicates between the physics model and the 3-D environment at each calculation timestep. The remainder of the data structures are set up and passed between the HVE interface and the physics before calculations begin.

What Am I?

In order to be completely general, as part of the set up process the physics sends HVE several data structures which tell HVE what services it requires from the interface. For example, a four-wheeled passenger car simulator requires that HVE allow the user to enter braking force at each wheel. These data must be entered by the user while setting up the event, so HVE must display a brake table dialog. Similarly, position, damage, payload and other dialogs must be displayed. The EventData structures pass this information from the calculation method to the HVE interface to tell HVE which dialogs to display.

For a detailed explanation of the HVE data structures, refer to Appendix A. Careful review of these structures is useful because these structures provide the exact definition for the HVE models and output variables.

Procedure For Developing HVE-compatible Simulations

The programmer produces an HVE-compatible human or vehicle dynamics program using the following steps:

- write a physics routine which includes control logic, a numerical integration method, a physical analysis of the human(s) and/or vehicle(s), derivative (acceleration) calculations and an output routine. Name this function ExecuteCalcMethod().
- write an input function which loads and parses the necessary HVE human, vehicle, environment, event and interface data. Name this function ParseHveInput().
- write a function which tells HVE which event dialogs output types to make available to the user. Name this function CalcMethodInfo().
- write a function which tells HVE which output variables the simulation produces, and whether an output variable may be user-edited. Name this function OutputTrackSetup().
- insert the SendHveOutput() function into the simulation's output routine.
- insert the GetSurfaceInfo() function into the physics routine where interaction forces between an object and its environment are calculated (e.g., the tire model).
- write a function which contains all the final results and warning messages. Name this function ShutDown().
- compile each source file.
- link each source file, including the hve.lib library in the link step.
- insert the resulting executable in the HVE /CalcMethods sub-directory.

When HVE is executed, the program name will appear in the list of available programs during Event Mode.

A complete sample program illustrating the above procedure is contained in Appendix B.

Discussion

The HVE Developer's Toolkit may be used by relatively simple, 2-D programs as well as sophisticated, 3-D programs. Although the HVE models contain literally hundreds of parameters per human or vehicle, the programmer may select as many or as few of these parameters as are necessary to meet the needs of the programmer's physics model.

An existing calculation method may be modified to become HVE-compatible using the procedure described above. The process basically involves stripping away the existing input and output routines, and replacing them with their HVE counterparts (refer to the procedure, above; also, refer to Appendix B).

The HVE Developer's Toolkit is written in C and assumes the calculation method will be programmed in C or C++. FORTRAN programs have not yet been included. It is, however, possible that FORTRAN programs could be incorporated using C or C++ "wrappers" around the FORTRAN program. This requires further research.

The HVE interface has been developed for use on Silicon Graphics (SGI) workstations, and leverages off the SGI graphics and video libraries. HVE requires a level of graphics processing power not yet available on personal computers (see [16] for performance comparisons). In the future, it is anticipated that HVE may be ported to other platforms, including Microsoft Windows NT™.

A program, called AUTOSIM™, is available from The Transportation Research Institute at the University of Michigan. AUTOSIM is a computer language designed to automatically generate simulation programs for the study of mechanical systems, such as vehicles [18]. Safety researchers with a background in dynamics and an elementary understanding of the C programming language may wish to use AUTOSIM to generate the C code for the simulation, and link this code into the HVE interface, using the procedures described in this paper. The combination of AUTOSIM and the HVE Developer's Toolkit provides a complete system for producing 2-D and 3-D vehicle simulations which include a robust, 3-D scientific visualization/animation user interface.

Summary

This paper has presented an overview of the HVE Developer's Toolkit, a library of functions and data structures which may be used by programmers to produce sophisticated, 3-D human and vehicle dynamics programs.

Use of the HVE Developer's Toolkit reduces the programmer's task by approximately 25 to 50 percent, while producing a significant improvement in program quality and usefulness.

Because of the visual 3-D output, use of the HVE interface as a development tool also greatly reduces the amount of time required to debug human and vehicle dynamics programs

Trademarks

HVE, EDVAP, EDCRASH, EDSVS, EDVTS, EDSMAC, EDHIS, EDVDS and EDVSM are trademarks of Engineering Dynamics Corporation. Windows NT is a trademark of Microsoft Corp. AUTOSIM is a trademark of the Regents of the University of Michigan

References

1. Robbins, D.H., Bennett, R.O., Roberts, V.L., "HSRI Three-Dimensional Crash Victim Simulator: Analysis, Verification,and User's Manual, and Pictorial Section", Report No. BIO M-70-9, HSRI, University of Michigan, June, 1971.

2. Bowman, B.M., Bennett, R.O., Robbins, D.H., "MVMA Two Dimensional Crash Victim Simulation, Version 4, Vol. I," Report No. UM-HSRI 79-5-1, University of Michigan, June, 1979.

3. Flect, J.T., Butler, F.E., and Vogel, SD.L., "An Improved Three Dimensional Computer Simulation of Crash Victims," NHTSA Report Nos. DOT-HS-801-507 through 510, April, 1975, Vols. 1-4.

4.Calspan Corporation, "Highway-Vehicle-Object Simulation Model", Calspan Report No. FHWA-RD-76-162, February, 1976.

5. McHenry, R.R., "Development of a Computer Program to Aid the Investigation of Highway Accidents," Calspan Report No. VJ-2979-V-1, DOT HS 800 821, December, 1971.

6. MacAdam, C.C., Fancher, P.S., Hu, G.T., Gillespie, T.D., "A Computerized Model for Simulating the Braking and Steering Dynamics of Trucks, Tractor-semi-trailers, Doubles, and Triples", Report No. UM-HSRI-80-58, HSRI, University of Michigan, June, 1980.

7. Moncartz, H.T., Bernard, J.E., Fancher, P.S., "A Simplified, Interactive Simulation for Predicting the Braking and Steering Response of Commercial Vehicles", Report No. UM-HSRI-PF-75-8, HSRI, University of Michigan, August, 1975.

8. Allen, R.W., Szostak, H.T., et al, "Analytical Modeling of Driver Response in Crash Avoidance Maneuvering: Vol. I - Technical Background," DOT-HS-807 270 NHTSA, Washington, DC, 1988.

9. Nalecz, A.G., "Development and Validation of Light Vehicle Dynamics Simulation (LVDS)," SAE Paper No. 920056, Society of Automotive Engineers, Warrendale, PA, 1992.

10. *Vehicle Analysis Package - EDSVS Program Manual, Version 4*, Engineering Dynamics Corporation, Beaverton, Oregon, 1993.

11. *Vehicle Analysis Package - EDVTS Program Manual, Version 4*, Engineering Dynamics Corporation, Beaverton, Oregon, 1993.

12. *Vehicle Analysis Package - EDCRASH Program Manual, Version 4*, Engineering Dynamics Corporation, Beaverton, Oregon, 1993.

13. *Vehicle Analysis Package - EDSMAC Program Manual, Version 2*, Engineering Dynamics Corporation, Beaverton, Oregon, 1993.

14. *EDC Simulations Training Manual, Version 4*, Engineering Dynamics Corporation, Beaverton, Oregon, 1989.

15. Lee, S.N., Fell, J.C., "An Historical Review of the National Highway Safety Administration's Field Accident Investigation Studies," NHTSA, Washington, DC, 1988.

16. Day, T.D., "A Computer Graphics Interface Specification for Studying Humans, Vehicles and Their Environment," SAE Paper No. 930103, Society of Automotive Engineers, Warrendale, PA, 1993.

17. *HVE Developer's Toolkit*, Engineering Dynamics Corporation, Beaverton, OR 1994 (currently under development).

18. *AUTOSIM Reference Manual, Version 2*, Transportation Research Institute, University of Michigan, Ann Arbor, MI, 1993.

Appendix A - HVE Data Structures

This appendix contains Tables 7 through 18, and documents each of the HVE data structures. Review of these structures reveals the characteristics of the human, vehicle and environment models, as well as the output variables for each human and vehicle

Table 7. HumanData Structure. Review of this structure provides an overview of the HVE human data model.

```
/* human.h
   Human Data Structure (12-1-93)
*/
     struct HumanData {
       long  Id;
       char  Name[MAXNAMELENGTH];
       SHORT  Location;
       SHORT  Sex;
       SHORT  Age;
       SHORT  BodyType;
       SHORT  Percentile;
       SHORT  NumSegments;
       SHORT  NumJoints;

       struct HumanTolerance {
         FLOAT  HIC;
         FLOAT  HeadPitch;
         FLOAT  HeadSideAccel;
         FLOAT  ChestSI;
         FLOAT  ChestForce;
         FLOAT  ChestAccel;
         FLOAT  KneeForce;
         FLOAT  LeftLap;
         FLOAT  LeftTorso;
         FLOAT  RightLap;
         FLOAT  RightTorso;
       } Tolerance;

       struct HumanSegment {
         struct SegmentInertia {
           FLOAT  Mass;
           FLOAT  Inertia[3];
           FLOAT  Weight;
           FLOAT  TotalWeight;
         } Inertia;

         SHORT  CurrentEllipsoid;
         SHORT  NumEllipsoids;    /* on this segment */

         struct SegmentEllipsoid {
           char  EllipsoidName[MAXNAMELENGTH];
           FLOAT  Coord[3];
           FLOAT  Length[3];
         } Ellipsoid[MAXELLIPSOIDS];

       } Segment[MAXSEGMENTS];

       struct HumanJoint {

         struct JointData {
           SHORT  SegNum;
           FLOAT  Coord[3]; /* rel to selected seg CG */
         } JointLoc[2];     /* proximal or distal seg */

         struct JointProperties {
           SHORT  Type;
           FLOAT  StopAngPlus[3];
           FLOAT  StopAngMinus[3];
           FLOAT  StopElasticityPlus[3];
           FLOAT  StopElasticityMinus[3];
           FLOAT  ElasticConst[3];
           FLOAT  DampingConst[3];
           FLOAT  ToleranceAnglePlus[3];
           FLOAT  ToleranceAngleMinus[3];
         } Property;
       } Joint[MAXJOINTS];
     };
/* end of human.h */
```

Table 8. VehicleData Structure. Review of this structure provides an overview of the HVE vehicle data model.

```
/* vehicle.h
   Vehicle Data structure (12-1-93)
*/
     struct VehicleData {
       long  Id;
       char  Name[MAXNAMELENGTH];
       SHORT  Type;
       char  Year[MAXNAMELENGTH];
       char  Make[MAXNAMELENGTH];
       char  Model[MAXNAMELENGTH];
       char  Style[MAXNAMELENGTH];
       char  ImageFilename[MAXNAMELENGTH];
       SHORT  NumAxles;
       SHORT  DriverSide;
       SHORT  EngineLocation;
       SHORT  DriveAxle;

       struct VehicleColor {
         FLOAT r;
         FLOAT g;
         FLOAT b;
       } Color;

       FLOAT ChangeCG[3];

       struct FrontDimension {
         FLOAT  Xf;
         FLOAT  FrontOverhang;
         FLOAT  OverallLength;
       } CGtoFront;

       struct RearDimension {
         FLOAT  Xr;
         FLOAT  RearOverhang;
         FLOAT  OverallLength;
       } CGtoRear;

       struct RightDimension {
         FLOAT  Yr;
         FLOAT  OverallWidth;
       } CGtoRight;

       struct LeftDimension {
         FLOAT  Yl;
         FLOAT  OverallWidth;
       } CGtoLeft;

       struct VehicleStiffness {
         FLOAT  Astf;
         FLOAT  Bstf;
         FLOAT  Kstf;
       } Stiffness[MAXSIDES];

       struct VehicleInertia {
         FLOAT  TotalWeight;
         FLOAT  TotalMass;
         FLOAT  SprungInertia[3];
       } Inertia;

       struct VehicleContactSurface {
         SHORT  NumSurfaces;
         SHORT  CurrentSurface;

         struct Surface {
           char  Name[MAXNAMELENGTH];
           SHORT  Location;  /* interior or exterior */
           FLOAT  Coord[MAXCORNERS][3];

           struct ContactMaterialProperty {
             char  MaterialName[MAXNAMELENGTH];
             FLOAT  LinStf;
             FLOAT  QuadStf;
             FLOAT  CubicStf;
             FLOAT  DampConst;
             FLOAT  PenetrnMax;
             FLOAT  ForceMax;
             FLOAT  EdgeConst;
             FLOAT  UnloadStf;
           } Property;
         } Surface[MAXCONTACTS];
       } Contact;

       struct VehicleBelt {
         SHORT CurrentLocation;

         struct BeltLocation {
           SHORT CurrentSection;

           struct BeltSection {
             BOOLEAN DeviceInstalled;

             struct BeltProperty {
               FLOAT Coord[3];  /* veh anchor pts */
               FLOAT  LinStf;
               FLOAT  QuadStf;
               FLOAT  CubicStf;
               FLOAT  DampConst;
               FLOAT  ForceMax;
               FLOAT  UnloadStf;
             } Property[2];     /* rt and lt sides */
           } Section[2];        /* Torso or Lap   */
         } Location[MAXBELTLOCATIONS]; /* 9 pos locs */
       } Belt;

       struct VehicleAirbag {
         SHORT CurrentLocation;

         struct AirbagProperties {
           BOOLEAN  DeviceInstalled;
           FLOAT  Coord[3];
           FLOAT  Radius;
           FLOAT  Length;
           FLOAT  Pressure;
           FLOAT  Thickness;
           FLOAT  Volume;
           FLOAT  VentCoef;
           FLOAT  VentArea;
           FLOAT  VentPress;
           FLOAT  DeflMax;
           FLOAT  ConvergCriterion;
           FLOAT  Elastic;
           FLOAT  ElasticReb;
           FLOAT  ElasticBotm;
           FLOAT  GasRho;
           FLOAT  GasFlowrate;
           FLOAT  GasCp;
           FLOAT  ColmDist;
           FLOAT  ColmLoad;
           FLOAT  ColmAngle;
           char  CurrentBackplane[MAXNAMELENGTH];
         } Property[MAXAIRBAGS];
       } Airbag;
```

Table 8. VehicleData Structure (cont).

```
struct VehicleConnection {

  struct FrontConnection {
    SHORT Type;
    FLOAT Coord[3];
  } Front;
  struct RearConnection {
    SHORT Type;
    FLOAT Coord[3];
    FLOAT Radius;
    FLOAT Friction;
    FLOAT ArticMax;
  } Rear;
} Connection;

struct VehicleDrag {
  FLOAT Cd;
  FLOAT LinearResist;
  FLOAT Const;
} Drag;

struct VehicleDrivetrain {
  struct EngineData {
    SHORT CurrentType;

    struct ThrottleStatus {
      SHORT TableLen;
      FLOAT Table[MAXENGINETABLE][3];                /*speed,power,torq*/
    } Status[2];        /* WOT, Closed   */
  } Engine;

  struct TransmissionData {
    SHORT CurrentTransType;
    FLOAT Ratio[MAXTRANSGEARS][MAXTRANSRATIOS];
  } TransData;

  struct DifferentalData {
    SHORT CurrentDiffType;
    FLOAT Ratio[MAXDIFFGEARS][MAXDIFFRATIOS];
  } DiffData;
} Drivetrain;

struct VehicleWheel {

  struct WheelLocation {
    FLOAT Coord[3];
  } Location;

  struct VehicleSuspension {
    SHORT CurrentSuspType;    /* ind, w-beam or 4-sprg */
    FLOAT InterTandemLoadXfer; /* per tandem axle set  */
    struct VehicleSpringShock {
      FLOAT LinearRate;
      FLOAT RollStf;          /* per axle       */
      FLOAT RollCtrHt;        /* per SOLID axle*/
      FLOAT LatSpringSpace;   /* per SOLID axle*/
      FLOAT DampRate;
      FLOAT Friction;
      FLOAT Hysteresis;
    } Spring;

    struct VehicleSuspensionInertia {
      FLOAT SolidAxleWeight;    /* per SOLID axle*/
      FLOAT SolidAxleMass;      /* per SOLID axle*/
      FLOAT SolidAxleInertia;   /*per SOLID axle*/
    } Inertia;

    struct VehicleDeflection {
      SHORT CurrentStop;        /* Upper or Lower */

      struct StopData {
        FLOAT MaxDeflection;
        FLOAT StopLinearRate;
        FLOAT StopCubicRate;
        FLOAT StopEnergyRatio;
      } Data[2];  /* Jounce, Rebound */
    } Deflect;

    struct VehicleSpindle {
      FLOAT Caster;
      FLOAT KingpinIncl;
      FLOAT Offset;
      FLOAT ToeIn;
    } Spindle;

    struct VehicleCamber {
      FLOAT Const;          /* Solid Axle */
      SHORT TableLen;
      FLOAT Data[MAXCAMBERTABLE][3];
              /* defl, camb, 1/2track */
    } Camber;

    struct VehicleAntiPitch { /*defl,AntiPitch*/
      SHORT TableLen;
      FLOAT Data[MAXCAMBERTABLE][2];
    } AntiPitch;     /* ind or solid axle */
    struct VehicleRollSteer {
      FLOAT Coef;         /* solid axle */
      FLOAT Const;        /* ind axle   */
      FLOAT Linear;
      FLOAT Quad;
      FLOAT Cubic;
    } RollSteer;
  } Suspension;

  struct VehicleTire {
    char  Name[MAXNAMELENGTH];
    char  Type[MAXNAMELENGTH];
    char  Mfr[MAXNAMELENGTH];
    char  Model[MAXNAMELENGTH];
    char  Size[MAXNAMELENGTH];

    BOOLEAN IsDual;
    FLOAT  DualSpace;

    struct PhysicalData {
      FLOAT UnloadedRadius;
      FLOAT InitialRideRate;
      FLOAT SecondRideRate;
      FLOAT SecondDefl;
      FLOAT MaxDefl;
      FLOAT PneumaticTrail;
      FLOAT AlignTorqCoef;
      FLOAT SpinInertia;
      FLOAT Weight;
      FLOAT Mass;
    } Physical;

    struct FrictionTable {
      SHORT NumTableLoads;
      FLOAT Load[MAXTIRETABLE];
      SHORT NumTableSpeeds;
      FLOAT Speed[MAXTIRETABLE];
      FLOAT InUse;

      struct MuData {     /* load and speed */
        FLOAT MuXPeak;
        FLOAT MuYPeak;
        FLOAT MuSlide;
        FLOAT PeakPcnt;
        FLOAT LongStiff;
      } Mu[MAXTIRETABLE][MAXTIRETABLE];
    } Friction;

    struct CalfaTable {
      SHORT NumTableLoads;
      FLOAT Load[MAXTIRETABLE];
      SHORT NumTableSpeeds;
      FLOAT Speed[MAXTIRETABLE];
      FLOAT InUse;
              /* load     speed */
      FLOAT Data[MAXTIRETABLE][MAXTIRETABLE];
    } Calfa;

    struct CgammaTable {
      SHORT NumTableLoads;
      FLOAT Load[MAXTIRETABLE];
      SHORT NumTableSpeeds;
      FLOAT Speed[MAXTIRETABLE];
      FLOAT InUse;
              /* load     speed */
      FLOAT Data[MAXTIRETABLE][MAXTIRETABLE];
    } Cgamma;

    struct RollOffTable {
      SHORT NumTableSlips;
      FLOAT Slip[MAXTIRETABLE];
      SHORT NumTableAngles;
      FLOAT Angle[MAXTIRETABLE];
      FLOAT InUse;

      struct RollOffData {
        FLOAT Long;
        FLOAT Lat;
            /* slip     angle */
      } Data[MAXTIRETABLE][MAXTIRETABLE];
    } RollOff;
  } Tire;

  struct VehicleWheelBrake {
    FLOAT LagTime;
    FLOAT RiseTime;
    FLOAT PushoutPress;
    FLOAT TorqueRatio;

    BOOLEAN IsProportion;
    FLOAT ProportionPress;
    FLOAT ProportionRatio;

    BOOLEAN IsAntilock;
    FLOAT  AntilockEffectiveness;
  } Brake;
} Wheel[MAXHVEAXLES][2]; /* right and left sides */

struct VehicleBrakeSystem {
  FLOAT PedalRatio;
} Brakes;

struct VehicleSteerSystem {
  SHORT CurrentAxle;

  struct SteerSystemData {
    BOOLEAN  IsSteerable;
    FLOAT    Ratio;
    FLOAT    ColumnStiffness;
    FLOAT    LinkageStiffness;
  } Data[MAXHVEAXLES];
} Steering;
};

/* End of vehicle.h */
```

Table 9. EnvironmentData Structure. Review of this structure provides an overview of the HVE environment data model.

```
/* environ.h
*/

/* EnvironmentData struct
*/
struct EnvironmentData {
  struct LocationData {
     FLOAT Latitude;
     FLOAT Longitude;
  } Data;

  char   Name[MAXNAMELENGTH];

  char   Date[MAXNAMELENGTH];
  SHORT  Time;
  FLOAT  AxisAngle;
  FLOAT  WindSpeed;
  FLOAT  WindDirection;
  FLOAT  AmbientPress;
  FLOAT  AmbientTemp;
  FLOAT  Overcast;
  FLOAT  VisibilityDist;
  FLOAT  Gravity;
};
/* End of Environ.h */
```

Table 10. EventData Structure. Review of this structure provides an overview of the HVE event data model.

```
/* event.h
     Event Data structure
*/
enum EventGeneralAnalysisOptions {ImpactVelocity,SeparationVelocity};
enum EventEdcrashOptions {Normal,TrajectorySim,SustainedContact};
enum EventPositions   {Initial,BegPerception,BegBraking,Impact,
                        Separation,PointOnCurve,EndOfRotation,Final};
enum EventStartMethod  {Euler,ModRungaKutta,RungaKutta,MAXSTARTMETH};
enum EventPredictorCorrector {AdamsMoulton,MilneHamming,MAXPREDCORRECT};
enum EventRestraintType {ShoulderBelt,LapBelt,Airbag,MAX_RESTRAINT_TYPES};
enum EventCalculationMethods
              {GeneralAnalysis,Edcrash,Edsmac,Edsvs,Edvts,
              Edhis,Edvds,Edvsm};

/* EventEdsmacOptions struct

    Special options for EDSMAC calculation method

*/
struct EventEdsmacOptions {
  FLOAT VectorSpacing;
  FLOAT VectorAdjustmentIncrement;
  FLOAT VectorForceTolerance;
  FLOAT IntervehicleFriction;
  FLOAT MinVelocityForFriction;
  FLOAT RestitutionConstant;
  FLOAT RestitutionLinearCoeff;
  FLOAT RestitutionQuadraticCoeff;
};

struct EventEdhisOptions {
     SHORT StartMethod;
     SHORT PredictorCorrectorMethod;
     FLOAT MinAcceleration;
     FLOAT VelocityChangeLimit;
     FLOAT AccelChangeLimit;
     FLOAT VelocityConvergeCriterion;
     FLOAT PrintTimeCriterion;

     struct EventIntegrationWeights {
          FLOAT TorsoFwd;
          FLOAT TorsoLat;
          FLOAT TorsoVert;
          FLOAT TorsoRoll;
          FLOAT TorsoPitch;
          FLOAT TorsoYaw;
          FLOAT HeadFwd;
          FLOAT HeadLat;
          FLOAT HeadVert;
          FLOAT LegRoll;
          FLOAT LegPitch;
          FLOAT LegYaw;
     } IntegrationWeights;
};

/* EventRestraintSystems struct
*/
struct EventRestraintSystems {
  enum EventRestraintType RestraintType;

  struct EventBeltData {
    BOOLEAN InUse;
    SHORT   SegmentBeltAttachedTo;
    FLOAT   SegmentBeltCoord[3];
    FLOAT   beltSlackLeft;
    FLOAT   beltSlackRight;
  } BeltData[MAXBELTTYPES];       /* MAXBELTTYPES = 2 */

  struct EventAirbagData {
    BOOLEAN InUse;
    FLOAT   BeginFillTime;
    FLOAT   FillDuration;
  } AirbagData;
};

/* EventSelectInteractions struct
*/
struct EventSelectInteractions {
  BOOLEAN interactions[MAXELLIPSOIDS][MAXCONTACTS];
};

/* EventInfo struct
*/
struct EventInfo {

  char  Name[MAXNAMELENGTH];
  int   NumSelectedHumans;
  long  SelectedHumanIDs[MAXHUMANS];
  int   NumSelectedVehicles;
  long  SelectedVehicleIDs[MAXVEHICLES];
  int   NumSelectedObjects;
  long  SelectedObjectIDs[MAXOBJECTS];          /* MAXHUMANS + MAXVEHICLES */
  enum  EventCalculationMethods CalcMethod;


  /* Calculation Options
  */
  enum  EventGeneralAnalysisOptions GaOptions;
  enum  EventEdcrashOptions         EdcrashOptions;
  struct EventEdsmacOptions         EdsmacOptions;
  struct EventEdhisOptions          EdhisOptions;
};

/* EventData struct
*/
struct EventData {
     BOOLEAN                        VerifyPosVel;
     long                           Id;
     struct EventInfo               Info;
     struct EventRestraintSystems   RestraintSystems;
     struct EventSelectInteractions SelectInteractions;
};

/*EventCalcMethodOptions struct

 Note: HVE needs to verify that the dialog pops up at all by ANDing all the grayed fields.
 This applies to the following dialogs:
        Payload, Throttle Table, Brakes Table, Steer Table, Restraints
*/
struct EventCalcMethodOptions {

  /* THESE VARIABLES DEFINE WHICH DIALOGS ARE SELECTABLE IN
     THE EDIT MENU, AND IF APPLICABLE, WHICH FIELDS IN
     DIALOGS ARE GRAY/UNGRAY.
  */
  BOOLEAN IsReconstruction;       /* IS METHOD RECONSTRUCTION*/
  BOOLEAN IsSimulation;           /* IS METHOD SIMULATION     */
  BOOLEAN ThreeDPosVel;           /* IN POSITION/VELOCITY DIALOG
                                     - IF FALSE, GRAY OUT
                                       Z,ROLL,PITCH AND USE
                                       AUTOPOSITION TO DECIDE
                                       THESE VALUES          */

  /* THESE ARE USED IN MAKING VEHICLES OPAQUE OR TRANSLUCENT
     WHILE POSITIONING
  */
  BOOLEAN InitialPosIsUsed;
  BOOLEAN BegPerceptionPosIsUsed;
  BOOLEAN BegBrakingPosIsUsed;
  BOOLEAN ImpactPosIsUsed;
  BOOLEAN SeparationPosIsUsed;
  BOOLEAN PointOnCurvePosIsUsed;
  BOOLEAN EndOfRotPosIsUsed;
  BOOLEAN FinalPosIsUsed;

  BOOLEAN HumansIsUsed;                /* METHOD USES HUMANS    */
  BOOLEAN VehicleIsUsed;               /* METHOD USES VEHICLES  */
  BOOLEAN DamageDataDlgIsUsed;         /* DISPLAY DAMAGE DATA DLG*/
  BOOLEAN PayloadXIsUsed;              /* PAYLOAD DLG- X REQ'D   */
  BOOLEAN PayloadYIsUsed;              /* PAYLOAD DLG- Y REQ'D   */
  BOOLEAN PayloadZIsUsed;              /* PAYLOAD DLG- Z REQ'D   */
  BOOLEAN PayloadRollIsUsed;           /* PAYLOAD DLG- ROLL REQ'D*/
  BOOLEAN PayloadPitchIsUsed;          /* PAYLOAD DLG- PITCH REQ'D*/
  BOOLEAN PayloadYawIsUsed;            /* PAYLOAD DLG- YAW REQ'D */
  BOOLEAN ThrottleWOTIsUsed;           /* THROTTLE DLG- WOT AVAIL */
  BOOLEAN ThrottleTractiveEffortIsUsed; /* THROTTLE DLG- TRACTIVE  */
  BOOLEAN ThrottleFrictionIsUsed;      /* THROTTLE DLG- FRICTION  */
  BOOLEAN BrakesPedalForceIsUsed;      /* BRAKES DLG-PEDAL FORCE */
  BOOLEAN BrakesWheelForceIsUsed;      /* BRAKES DLG-WHEEL FORCE*/
  BOOLEAN BrakesFrictionIsUsed;        /* BRAKES DLG- FRICTION    */
  BOOLEAN WheelDataDlgIsUsed;          /* WHEEL DATA DLG AVAIL    */
  BOOLEAN GearTableDlgIsUsed;          /* GEAR TABLE DLG AVAIL    */
  BOOLEAN SteerAtWheelsIsUsed;         /* STEER DLG - @ STRG      */
  BOOLEAN SteerAtTiresIsUsed;          /* STEER DLG - @ TIRES     */
  BOOLEAN CollisionPulseDlgIsUsed;     /* COLLISION PULSE DLG     */
  BOOLEAN ProducesCollisionPulse;      /* PRODUCE A COLSN PULSE   */
  BOOLEAN BeltRestraintsIsUsed;        /* RESTRAINTS OPTIONS      */
  BOOLEAN AirbagRestraintsIsUsed;      /* RESTRAINTS OPTIONS      */
  BOOLEAN ContactsDlgIsUsed;           /* CONTACTS IN EDIT MENU   */

};
/* EventCalcMethodOutputType struct

  Define the possible output types available from a Calc
  Method.

*/
struct EventCalcMethodOutputType {

  /* THESE VARIABLES DEFINE THE TYPES OF OUTPUT AVAILABLE IN
     PLAYBACK
  */
  BOOLEAN AccidentHistory;
  BOOLEAN DamageData;
  BOOLEAN DamageProfiles;
  BOOLEAN DataGraphing;
  BOOLEAN HumanData;
  BOOLEAN InjuryData;
```

Table 10. EventData Structure (cont.).

```
    BOOLEAN Messages;
    BOOLEAN MomDiagramDamage;
    BOOLEAN MomDiagramScene;
    BOOLEAN ProgramData;
    BOOLEAN Results;
    BOOLEAN SiteDrawing;
    BOOLEAN TrajSimulation;
    BOOLEAN VariableOutput;
    BOOLEAN VehicleData;
};

/*
    Name:     EventObjectHeader

    Purpose:  This structure is filled out by the calc
              method using EventInfo struct. See
              documentation in EventFromCalcMethodHeader
              for more details.

*/
struct EventObjectHeader {

   long ObjectID;

   /* IDs FOR THE DATA FOLLOWING ALL THE HEADERS
      WHEN A -1L IS REACHED, END OF ATTACHMENTS
   */
   long WhichIDs[MAXATTACHEDOBJECTS];

   /* -1L=ENVIRONMENT, ELSE, ID OF THE OBJECT'S COORD
      SYSTEM THIS OBJECT IS RELATIVE TO
   */
   long RelativeCoordSystemID;

};

/*
    Name:     EventFromCalcMethodHeader

    Purpose:  Return header from the calculation method.
              The calculation method must fill out the
              EventCalcMethodOptions structure so we know which fields to
              gray/ungray. It also must decipher from EventInfo which objects
              are allowed and if their connections are valid. In doing this,
              it fills out the NumObjects field and object array.

              NumObjects = the number of objects, NOT
              including their attachments, for example,
              if we have 2 semi-trailers, NumObjects = 2 and
              object[0].objectID    = truck 1 ID
              object[0].whichIDs[0] = truck 1's trailer 1
              object[0].whichIDs[1] = truck 1's dolly for trailer 2
              object[0].whichIDs[2] = truck 1's trailer 2
              object[0].whichIDs[3] = truck 1's dolly for trailer 3
              object[0].whichIDs[4] = truck 1's trailer 3

              object[1].objectID    = truck 2 ID
              object[1].whichIDs[0] = truck 2's trailer 1
              object[1].whichIDs[1] = truck 2's dolly for trailer 2
              object[1].whichIDs[2] = truck 2's trailer 2
              object[1].whichIDs[3] = truck 2's dolly for trailer 3
              object[1].whichIDs[4] = truck 2's trailer 3

              Then, the Calc Control Panel can fill out the selected listbox appropriately.
*/
struct EventFromCalcMethodHeader {
  struct EventCalcMethodOptions     Options;
  SHORT                             NumObjects;
  struct EventObjectHeader          Object[MAXOBJECTS];
  struct EventCalcMethodOutputType  OutputType;
};
```

Table 11. EventHumanData Structure. Review of this structure provides an overview of the HVE event-related parameters pertaining to humans.

```
/* evnthum.h
*/
/* EventHumanPosVel struct
*/
struct EventHumanPosVel {

   struct EventHuPosVelData {

     /* Position
     */
     BOOLEAN PosnIsUsed;
     FLOAT xPos;
     FLOAT yPos;
     FLOAT zPos;
     FLOAT rollOrient;
     FLOAT pitchOrient;
     FLOAT yawOrient;

     /* Velocity
     */
     BOOLEAN VelocityIsUsed;
     FLOAT  fwdVel;
     FLOAT  latVel;
     FLOAT  vertVel;
     FLOAT  rollVel;
     FLOAT  pitchVel;
     FLOAT  yawVel;

   } Data[MAXSEGMENTS][MAXPOSITIONS]; /* MAXSEGMENTS=3, MAXPOSITIONS=8*/
};

/* EventHumanData struct

   This struct holds data specific to each human in each
   event.
*/
struct EventHumanData {
   long            Id;    /* WHICH HUMAN'S EVENT DATA? */
   struct EventHumanPosVel  PosVel;
};
```

Table 12. EventVehicleData Structure. Review of this structure provides an overview of the HVE event-related parameters pertaining to humans.

```
/* evntveh.h
*/
   enum DamageBasis {EBS,DamageProfile,MAX_BASE_TYPES=2};

/* EventVehiclePosVel struct
*/
struct EventVehiclePosVel {

   struct EventVePosVelData {

     /* Position
     */
     BOOLEAN PosnIsUsed;
     FLOAT XPos;
     FLOAT YPos;
     FLOAT ZPos;
     FLOAT RollOrient;
     FLOAT PitchOrient;
     FLOAT YawOrient;

     /* Velociy
     */
     BOOLEAN VelocityIsUsed;
     FLOAT  uVel;
     FLOAT  vVel;
     FLOAT  wVel;
     FLOAT  SlipAngle;
     FLOAT  TotalVel;
     FLOAT  RollVel;
     FLOAT  PitchVel;
     FLOAT  YawVel;

   } Data[MAXPOSITIONS]; /* MAXPOSITIONS=8    */
};

/* EventVehicleDamageData struct - used by reconstruction calculation methods only
*/
struct EventVehicleDamageData {


   BOOLEAN ValidCDC;                /* NEED BOOLEAN, SINCE TESTING FOR
                                         'None' IS NOT LANGUAGE INDEPENDENT */
   char   Cdc[MAXCDCLENGTH];        /* INITIALLY, 'None'                    */
   FLOAT  Pdof;
   FLOAT  ImpulseCenterX;
   FLOAT  ImpulseCenterY;

   /* THE BASIS DETERMINES WHETHER TO USE ebs OR THE Damage
      Profile DATA
   */
   enum DamageBasis Basis;

   /* FOR EBS - SAME AS DeltaVtot
   */
   FLOAT DeltaVtot;
   FLOAT DeltaVFwd;
   FLOAT DeltaVLat;
   FLOAT DeltaVAng;        /* FOR DAMAGE PROFILE*/
   int NumZones;
   FLOAT Width;
   FLOAT Offset;
   FLOAT CrushDepths[MAXCRUSHENTRIES]; /* MAXCRUSHENTRIES = 10 */

   struct EventAStiffnessCoeffs {
     FLOAT Coef[MAXCRUSHZONES];
   } AStiffness;

   struct EventBStiffnessCoeffs {
     FLOAT Coef[MAXCRUSHZONES];
   } BStiffness;

   /* ADDITIONAL OUTPUTS
   */
   FLOAT Energy;
   FLOAT Force;
};

/* EventPayloadData struct
*/
struct EventPayloadData {
   BOOLEAN Exists;
   FLOAT  Coord[3];
   FLOAT  Weight;
   FLOAT  Mass;
   FLOAT  Inertia[3];
};


/* EventLights struct
*/
struct EventLights {

   BOOLEAN IsOn[MAXLIGHTS];
};
```

Table 10. EventVehicleData Structure (cont.).

```
/* EventThrottleTable struct
*/
struct EventThrottleTable {

   SHORT ThrottleOption;

   struct ThrottleTableData {
      SHORT TableLength;

      struct {
         FLOAT Time;
         FLOAT Table[MAXHVEAXLES][2];
         FLOAT Value;
      } Data[MAXDRIVERTABLE];

   } ThrottleData[MAXDRIVERTABLETYPES];

};

/* EventBrakeTable struct
*/
struct EventBrakeTable {

   SHORT BrakeOption;

   struct BrakeTableData {
      SHORT TableLength;

      struct {
         FLOAT Time;
         FLOAT Table[MAXHVEAXLES][2];
         FLOAT Value;
      } Data[MAXDRIVERTABLE];

   } BrakeData[MAXDRIVERTABLETYPES];

};

/* EventGearTable struct
*/
struct EventGearTable {

   SHORT GearBoxOption;

   /* THIS IS FOR Transmission OPTION
   */
   SHORT NumTransShifts;

   struct TransShiftData {
     FLOAT   Time;
     SHORT   WhichGear;              /* see VeTransmissionData   */
   } TransData[MAXGEARTABLE];        /* MAXGEARTABLE = 10        */

   /* THIS IS FOR Differential OPTION
   */
   SHORT NumDiffShifts;
   struct DiffShiftData {
     FLOAT   Time;
     SHORT   WhichGear;   /* see VeDifferentialData   */
   } DiffData[MAXGEARTABLE]; /* MAXGEARTABLE = 10       */

};

/* EventSteerTable struct
*/
struct EventSteerTable {
   SHORT SteerOption;
   struct SteerTableData {
      SHORT TableLength;

      struct {
         FLOAT Time;
         FLOAT Table[MAXHVEAXLES][2];
         FLOAT Value;
      } Data[MAXDRIVERTABLE];

   } SteerData[MAXDRIVERTABLETYPES];

};

/* EventWheelData struct
*/
struct EventWheelData {

   FLOAT  DragFactor;
   BOOLEAN IsRotLatSkidding;

   struct EventWheelLockupSteerData {
     FLOAT WheelLockup;
     FLOAT WheelSteer;
   } Data[MAXHVEAXLES][2];

};

/* EventDriverControls struct
*/
struct EventDriverControls {
   struct EventThrottleTable         ThrottleTable;
   struct EventBrakeTable            BrakeTable;
   struct EventGearTable             GearTable;
   struct EventSteerTable            SteerTable;
   struct EventWheelData             WheelData;
};
```

```
/* EventCollisionPulse struct
*/
struct EventCollisionPulse {

  long calcID;          /* which calc the data was gotten from
                           vehicle must be in this calc method
                           to select this.
                           -1L = USER-ENTERED                          */

  char filename[MAXFILENAMELENGTH];        /* filename the data was
                                              gotten from or saved to.
                                              "" - MEANS NOT SAVED
                                              OR GOTTEN FROM A FILE*/

  SHORT TableLength;
  struct EventPulseAccelData {
    FLOAT time;
    FLOAT forward;
    FLOAT lateral;
    FLOAT vertical;
    FLOAT roll;
    FLOAT pitch;
    FLOAT yaw;
  } PulseAccel[MAXPULSTABLELEN]; /* MAXPULSTABLELEN = 100*/

  struct EventInUsePulseFactors {
    FLOAT fwd;
    FLOAT lateral;
    FLOAT vertical;
    FLOAT roll;
    FLOAT pitch;
    FLOAT yaw;
  } PulseFactors;
};

/* EventVehicleData  struct

  This struct holds data specific to each vehicle in each event.
*/
struct EventVehicleData {
  long                              Id;        /* WHICH VEHICLE'S EVENT DATA?  */
  struct EventVehiclePosVel         PosVel;
  struct EventVehicleDamageData     VehicleDamage;
  struct EventPayloadData           Payload;
  struct EventLights                Lights;
  struct EventDriverControls        DriverControls;
  struct EventCollisionPulse        CollisionPulse;
};
```

Table 13. InterfaceData Structure. Review of this structure provides an overview of the HVE interface variables, used mostly by simulations.

```
/* intrface.h
*/

enum PlaybackType {Storage,Animation};

/* InterfaceData struct
*/
struct InterfaceData {

  /* Integration Timesteps
  */
  FLOAT   dtHumanCol;
  FLOAT   dtVehicleCol;
  FLOAT   dtCurbCol;
  FLOAT   dtVehSep;
  FLOAT   dtVehTraj;
  FLOAT   dtOutput;

  /* Termination Conditions
  */
  FLOAT   Tmax;
  FLOAT   TermLinearVel;
  FLOAT   TermAngularVel;
  SHORT   MaxBisections;
  FLOAT   VelocityConvergence;
  FLOAT   VelocityChangeLimit;
  FLOAT   AccelerationChangeLimit;

  /* Playback
  */
  enum PlaybackType  PlaybackMode;
  FLOAT       dtPlayback;
};
```

Table 14. The following portion of output.h shows each of the output variables monitored by HVE.

```
/* output.h

  Output data structures and variable definitions.

*/
/* VEHICLE OUTPUT DATA GROUPS (for reference only)
  SMKinematics (sprung mass kinematics)
  SMKinetics (sprung mass kinetics)
  TireData (locn, forces/moments, driver inputs, skid/scuff)
  WheelData (kinematics, forces, driver inputs)
  Connection (relative angles, forces and moments)
  Drivetrain (engine, transmission, differential
  Driver (throttle, brake, gear, steering)
```

Table 14. HVE Outputs (cont.)

```
Below are the definitions for each data group (for reference only)

    Sprung Mass Kinematics Group (pos, vel, accel)

    position
    SMKinematics[0] =  G X coord
    SMKinematics[1] =  CG Y coord
    SMKinematics[2] =  CG Z coord
    SMKinematics[3] =  roll about x axis
    SMKinematics[4] =  pitch about y axis
    SMKinematics[5] =  yaw about z axis
    SMKinematics[6] =  path radius
    SMKinematics[7] =  course angle

    velocity
    SMKinematics[8] =  total velocity
    SMKinematics[9] =  sideslip angle
    SMKinematics[10] = longitudinal vel (u)
    SMKinematics[11] = side velocity (v)
    SMKinematics[12] = normal velocity (w or vertical)
    SMKinematics[13] = forward vel
    SMKinematics[14] = lateral velocity
    SMKinematics[15] = roll velocity (p)
    SMKinematics[16] = pitch velocity (q)
    SMKinematics[17] = yaw velocity (r)

    acceleration
    SMKinematics[18] = total acceleration
    SMKinematics[19] = long acceleration (udot)
    SMKinematics[20] = side acceleration (vdot)
    SMKinematics[21] = normal acceleration (wdot)
    SMKinematics[22] = forward acceleration
    SMKinematics[23] = lateral acceleration
    SMKinematics[24] = tangential accel
    SMKinematics[25] = centripetal accel
    SMKinematics[26] = roll acceleration (pdot)
    SMKinematics[27] = pitch acceleration (qdot)
    SMKinematics[28] = yaw acceleration (rdot)

    Sprung Mass Kinetics Group (forces & moments)
    SMKinetics[0] =    sum Fx from tires (suspension)
    SMKinetics[1] =    sum Fy from tires (suspension)
    SMKinetics[2] =    sum Fz from tires (suspension)
    SMKinetics[3] =    sum Mx from tires (suspension)
    SMKinetics[4] =    sum My from tires (suspension)
    SMKinetics[5] =    sum Mz from tires (suspension)
    SMKinetics[6] =    sum Fx from collision
    SMKinetics[7] =    sum Fy from collision
    SMKinetics[8] =    sum Fz from collision
    SMKinetics[9] =    sum Mx from collision
    SMKinetics[10] =   sum My from collision
    SMKinetics[11] =   sum Mz from collision
    SMKinetics[12] =   sum Fx from aerodynamics
    SMKinetics[13] =   sum Fy from aerodynamics
    SMKinetics[14] =   sum Fz from aerodynamics
    SMKinetics[15] =   sum Mx from aerodynamics
    SMKinetics[16] =   sum My from aerodynamics
    SMKinetics[17] =   sum Mz from aerodynamics
    SMKinetics[18] =   sum Fx from connection
    SMKinetics[19] =   sum Fy from connection
    SMKinetics[20] =   sum Fz from connection
    SMKinetics[21] =   sum Mx from connection
    SMKinetics[22] =   sum My from connection
    SMKinetics[23] =   sum Mz from connection

    Tire Group (tire location, forces and moments)
    NOTE:  ' indicates tire reference frame which has its
           origin at the center of the tire contact patch. x' axis
           is at angle, delta, relative to
           the vehicle x axis. z' is normal to the road
           plane and y' is orthogonal to x' and z'.
    TireData[MAXAXLES][2][0] =   x contact patch ctr
    TireData[MAXAXLES][2][1] =   y contact patch ctr
    TireData[MAXAXLES][2][2] =   z contact patch ctr
    TireData[MAXAXLES][2][3] =   X contact patch ctr
    TireData[MAXAXLES][2][4] =   Y contact patch ctr
    TireData[MAXAXLES][2][5] =   Z contact patch ctr
    TireData[MAXAXLES][2][6] =   long tire force (Fx')
    TireData[MAXAXLES][2][7] =   lat tire force (Fy')
    TireData[MAXAXLES][2][8] =   normal tire force (Fz')
    TireData[MAXAXLES][2][9] =   overturning mom (Mx')
    TireData[MAXAXLES][2][10] =  roll res moment (My')
    TireData[MAXAXLES][2][11] =  aligning torque (Mz')
    TireData[MAXAXLES][2][12] =  net wheel torque
    TireData[MAXAXLES][2][13] =  loaded tire radius
    TireData[MAXAXLES][2][14] =  camber angle of wheel
    TireData[MAXAXLES][2][15] =  slip angle (alpha)
    TireData[MAXAXLES][2][16] =  skid flag
    TireData[MAXAXLES][2][17] =  scuff flag

    Wheel Group (deflection, forces)
    position
    WheelData[MAXAXLES][2][0] =  x coord of wheel ctr
    WheelData[MAXAXLES][2][1] =  y coord of wheel ctr
    WheelData[MAXAXLES][2][2] =  z coord of wheel ctr
    WheelData[MAXAXLES][2][3] =  spin angle of wheel

    velocity
    WheelData[MAXAXLES][2][4] =  norm vel of wheel ctr
    WheelData[MAXAXLES][2][5] =  gamma-dot
    WheelData[MAXAXLES][2][6] =  spin velocity of wheel

    acceleration
    WheelData[MAXAXLES][2][7] =  norm accel of wheel ctr
    WheelData[MAXAXLES][2][8] =  gamma-ddot
    WheelData[MAXAXLES][2][9] =  spin acceleration of wheel

    forces
    WheelData[MAXAXLES][2][10] = long force at ctr (Fx)
    WheelData[MAXAXLES][2][11] = lat force at ctr (Fy)
    WheelData[MAXAXLES][2][12] = norm force at ctr (Fz)

    suspension deflection and forces
    WheelData[MAXAXLES][2][13] = wheel defl from equil
    WheelData[MAXAXLES][2][14] = wheel defl rate
    WheelData[MAXAXLES][2][15] = Fz at wheel from susp
    WheelData[MAXAXLES][2][16] = Fz at wheel from damp
    WheelData[MAXAXLES][2][17] = Fz at wheel from anti-pitch

    Results of driver controls (attempted throttle and
    brake torque at wheel, steer angle at wheel)
    WheelData[MAXAXLES][2][18] = drive torque, spin axis
    WheelData[MAXAXLES][2][19] = brake torque, spin axis
    WheelData[MAXAXLES][2][20] = brake pressure at wheel
    WheelData[MAXAXLES][2][21] = steer angle (delta)

    Sprung Mass Connections Group (position, orientation,
    forces & moments for an articulated object)
    Note that each vehicle can be towed by only one
    vehicle; thus, only one set of data is required
    per towed vehicle!
    Angles are relative to tow vehicle

    orientation
    Connection[0] =      roll articulation
    Connection[1] =      pitch articulation
    Connection[2] =      yaw articulation

    velocity
    Connection[3] = ang veloc about roll axis
    Connection[4] = ang veloc about pitch axis
    Connection[5] = ang veloc about yaw axis

    acceleration
    Connection[6] = ang accel about roll axis
    Connection[7] = ang accel about pitch axis
    Connection[8] = ang accel about yaw axis

    connection forces and moments
    Connection[9] =   Fx from connection
    Connection[10] = Fy from connection
    Connection[11] = Fz from connection
    Connection[12] = Mx from connection
    Connection[13] = My from connection
    Connection[14] = Mz from connection

    Drivetrain Group (engine, transmission, differential)
    engine
    Drivetrain[0]  =  engine speed
    Drivetrain[1]  =  engine power
    Drivetrain[2]  =  engine torque

    transmission and differential gear ratios
    Drivetrain[3]  =  transmission gear ratio
    Drivetrain[4]  =  differential gear ratio

    Driver Controls Group (throttle, brake, gear,
    steering)
    Driver[0]     =  throttle position
    Driver[1]     =  brake pedal force
    Driver[2]     =  brake system pressure
    Driver[3]     =  transmission gear number
    Driver[4]     =  differential gear number
    Driver[5]     =  steer angle at wheel

*/

/* Output track info structure for each variable
*/
struct OutputTrackVariable {
    char   ResName[MAXNAMELENGTH];
    SHORT  Status;       /* EDITABLE, NOT_EDITABLE, NOT_USED */
};
```

Table 15. HumanOutputTrackSetup Structure. Review of this structure provides an overview of the HVE human output data model.

```
/* Setup structure for each human output track variable
*/
struct HumanOutputTrackSetup {
    long  Id;
    struct OutputTrackVariable SMKinematics[MAX_MASS_KINEMATIC_VARS];
    struct OutputTrackVariable SMKinetics[MAX_MASS_KINETIC_VARS];
};
```

Table 16. OutputHumanData Structure. Review of this structure provides an overview of the HVE human output data model.

```
/* Output structure for human at each timestep
*/

struct OutputHumanData {
    long  Id;
    FLOAT SMKinematics[MAX_MASS_KINEMATIC_VARS];
    FLOAT SMKinetics[MAX_MASS_KINETIC_VARS];
};
```

Table 17. VehicleOutputTrackSetup Structure. Review of this structure provides an overview of the HVE vehicle output data model.

```
/* Setup structure for each vehicle output track variable
*/
struct VehicleOutputTrackSetup {
    long Id;
    struct OutputTrackVariable Kinematics[MAX_MASS_KINEMATIC_VARS];
    struct OutputTrackVariable SMKinetics[MAX_MASS_KINETIC_VARS];
    struct OutputTrackVariable TireData[MAX_TIRE_VARS];
    struct OutputTrackVariable WheelData[MAX_WHEEL_VARS];
    struct OutputTrackVariable Connection[MAX_CONNECTION_VARS];
    struct OutputTrackVariable Drivetrain[MAX_DRIVETRAIN_VARS];
    struct OutputTrackVariable Driver[MAX_DRIVER_VARS];
};
```

Table 18. OutputVehicleData Structure. Review of this structure provides an overview of the HVE vehicle output data model.

```
/* Output track data structure (for each output time increment).
*/
struct OutputVehicleData {
    long Id;
    FLOAT SMKinematics[MAX_MASS_KINEMATIC_VARS];
    FLOAT SMKinetics[MAX_MASS_KINETIC_VARS];
    FLOAT TireData[MAX_TIRE_VARS];
    FLOAT WheelData[MAX_WHEEL_VARS];
    FLOAT Connection[MAX_CONNECTION_VARS];
    FLOAT Drivetrain[MAX_DRIVETRAIN_VARS];
    FLOAT Driver[MAX_DRIVER_VARS];
};


/* end of output.h */
```

Appendix B - Sample HVE-Compatible Program

The following illustrates an example of an HVE-compatible program, written in C. This simple 2-D program accelerates a vehicle from its initial position and velocity until the user-specified maximum simulation time is reached. A brief explanation for each function is provided.

All functions for this program are contained in a single file, named sample.c. The file begins by naming all the functions and a list of header files to be included (with the exception of math.h, these files are suppliedby the HVE Developer's Toolkit). Also, global HVE data structures are declared.

```
/* FILE: sample.c

Sample program which illustrates use of HVE interface.

        Functions included:
            ExecuteCalcMethod()
            ParseHveInput()
            CalcMethodInfo()
            SelectOutputTrackVars()
            ShutDown()
*/

#include <math.h>

/* HVE definitions and headers
*/
#include "..\lib\hvedef.h"
#include "..\lib\angles.h"
#define RIGHT           0
#define LEFT            1

/* HVE data structure headers
*/
#include "..\lib\vehicle.h"
#include "..\lib\evntveh.h"
#include "..\lib\environ.h"
#include "..\lib\event.h"
#include "..\lib\intrface.h"
#include "..\lib\output.h"

/* HVE data structures
*/
struct VehicleData *Vehicle;
struct EventVehicleData *EventVehicle;
struct EnvironmentData Environment;
struct InterfaceData Interface;
struct EventFromCalcMethodHeader CalcMethodHeader;
struct VehicleOutputTrackSetup *VehicleOutputTrack;
struct OutputVehicleData OutputVehicle;
```

Next, the user-defined functions and variables used by sample.c are declared (this simple program declares most of its variables as global).

```
/* Prototypes for user-defined functions
*/
SHORT WriteOutput(void);

/*------------------ GLOBAL VARIABLES ---------------------------*/
FLOAT a;                  /* vehicle forward acceleration (in/sec^2) */
FLOAT g;                  /* acceleration of gravity (in/sec^2)      */
FLOAT t;                  /* time (sec)                              */
FLOAT tmax;               /* maximum simulation time (sec)           */
FLOAT dtprint;            /* print time interval (sec)               */
FLOAT V;                  /* current velocity (in/sec)               */
FLOAT V0;                 /* initial velocity (in/sec)               */
FLOAT vmin;               /* termination velocity (in/sec)           */
FLOAT CG_X;               /* X coordinate of CG at each time step (in)*/
FLOAT CG_X0;              /* initial X coordinate of CG (in)         */
FLOAT CG_Y;               /* Y coordinate of CG at each time step (in)*/
FLOAT CG_Z;               /* Z coordinate of CG at each time step (in)*/
FLOAT Roll;               /* roll angle of veh at each timestep (rad) */
FLOAT Pitch;              /* pitch angle of veh at each timestep (rad)*/
FLOAT Yaw;                /* yaw angle of veh at each timestep (rad) */
FLOAT x[4];               /* Wheel x-coord (in)                      */
FLOAT y[4];               /* Wheel y-coord (in)                      */
FLOAT z[4];               /* Wheel z-coord (in)                      */
FLOAT ThrottleMethod;     /* Type of HVE throttle table              */
FLOAT ThrottleTableLen;   /* Length of HVE throttle table            */
FLOAT BrakeMethod;        /* Type of HVE brake table                 */
FLOAT BrakeTableLen;      /* Length of HVE brake table               */
FLOAT SteerMethod;        /* Type of HVE steer table                 */
FLOAT SteerTableLen;      /* Length of HVE steer table               */
```

The following function is called by HVE to execute the simulation. This is the main physics program written by the user (it must be named ExecuteCalcMethod).

```
SHORT ExecuteCalcMethod()
{
    /* Function that executes the sample physics program.

      Called by:        HveMain
      Function Calls:   WriteOutput()
    */

    /* This code controls the execution of the calculation method.
    */
    do
    {
      a     = -.2*g;     /* constant! */
      V     = V0 + a*t;
      CG_X = CG_X0 + V0*t + 0.5*a*t*t;
      WriteOutput();
    } while (t < tmax + dtprint && V > vmin);

    return 0;
} /* End of ExecuteCalcMethod() */
```

The following function is called by HVE to assign all variables required by sample.c. It is supplied by the user, and must always be named ParseHveInput.)

```
/* Function: ParseHveInput()    Version 1.0 Source Code Listing
      Copyright 1993, Engineering Dynamics Corporation
      All Rights Reserved.
*/

SHORT ParseHveInput(void)
{
    /* Assigns input data used by the sample program.

      Called by:        HveMain()
      Function Calls:   (none)

------------------------ LOCAL VARIABLES ---------------------------*/
SHORT WheelNum;         /* Local index                            */
SHORT MaxWheels;        /* Number of wheels on vehicle            */
SHORT AxleNum;          /* Local index                            */
SHORT MaxAxles;         /* Number of axles on vehicle             */
SHORT Side;             /* Local index                            */
/*----------------------------------------------------------------*/

    /* First, parse the environment data.
    */
    g = Environment.Gravity;

    /* Parse the general vehicle characteristics. Our sample
      program only needs wheel positions.
    */
    MaxAxles  = Vehicle[0].NumAxles;
    MaxWheels = 2*MaxAxles;
    for (WheelNum=0; WheelNumaxWheels; WheelNum++)
    {
      AxleNum = WheelNum/2;
      Side = (WheelNum % 2 == 0) ? RIGHT : LEFT;
      x[WheelNum] =Vehicle[0].Wheel[AxleNum][Side].Location.Coord[0];
      y[WheelNum] =Vehicle[0].Wheel[AxleNum][Side].Location.Coord[1];
      z[WheelNum] =Vehicle[0].Wheel[AxleNum][Side].Location.Coord[2];
    }

    /* Parse event-related vehicle data. Start with positions (the
      sample program has only one degree of freedom - X direction)
    */
    CG_X   =CG_X0 =  EventVehicle[0].PosVel.Data[0].XPos;
    CG_Y   = 0.0;
    CG_Z   = fabs(EventVehicle[0].PosVel.Data[0].ZPos);
    Roll   = 0.0;
    Pitch  = 0.0;
    Yaw    = 0.0;

    /* velocities
    */
    V = V0   = EventVehicle[0].PosVel.Data[0].uVel;

    /* Assign interface variables.
    */
    tmax     = Interface.Tmax;
    dtprint  = Interface.dtOutput;
    vmin     = Interface.TermLinearVel;
```

```
     /* Turn off driver controls (throttle, braking and steering
        tables).
     */
     ThrottleMethod      = EventVehicle[0].DriverControls.ThrottleTable.ThrottleOption;
     ThrottleTableLen    = 0;

     BrakeMethod         = EventVehicle[0].DriverControls.BrakeTable.BrakeOption;
     BrakeTableLen       = 0;

     SteerMethod         = EventVehicle[0].DriverControls.SteerTable.SteerOption;
     SteerTableLen       = 0;

     return 0;

} /* End of ParseHveInput() */
```

The following function is called by HVE, and tells HVE what kind of program it is, and what kinds of output it will produce. This function is user-supplied, and must be named CalcMethodInfo.

```
/* Function: CalcMethodInfo()   Version 1.0 Source Code Listing
      Copyright 1993, Engineering Dynamics Corporation
      All Rights Reserved.
*/

SHORT CalcMethodInfo(void)
{
     /* Sets up the truth table for HVE edit options.
     */
/*------------- LOCAL VARIABLES ---------------------*/
SHORT j;                  /* Local index              */
/*-------------------------------------------------*/

     CalcMethodHeader.NumObjects               = 1;

     CalcMethodHeader.Options.IsReconstruction  = FALSE;
     CalcMethodHeader.Options.IsSimulation      = TRUE;

     /* If the following is FALSE, do not allow user to enter
        positions for Z, roll and pitch. Instead, use Autoposition
        to compute the values.
     */
     CalcMethodHeader.Options.ThreeDPosVel      = FALSE;

     /* The following are used by HVE to decide whether vehicles are
        shown as translucent 'targets' (not used in calculations) or
        as normally rendered vehicles (used in calculations).
     */
     CalcMethodHeader.Options.InitialPosIsUsed  = TRUE;

     /* The following tell HVE to make the following output dialogs
        available in Playback mode.
     */
     CalcMethodHeader.OutputType.AccidentHistory = TRUE;
     CalcMethodHeader.OutputType.DataGraphing    = TRUE;
     CalcMethodHeader.OutputType.Messages        = TRUE;
     CalcMethodHeader.OutputType.ProgramData     = TRUE;
     CalcMethodHeader.OutputType.TrajSimulation  = TRUE;
     CalcMethodHeader.OutputType.VariableOutput  = TRUE;
     CalcMethodHeader.OutputType.VehicleData     = TRUE;

     CalcMethodHeader.Object[0].ObjectID         = 1;

     for (j = 0; jalcMethodHeader.NumObjects-1; j++)
     {
       CalcMethodHeader.Object[0].WhichIDs[j]    = 1;
     }
     CalcMethodHeader.Object[0].RelativeCoordSystemID    = -1L;

     return 0;

} /* End of CalcMethodInfo() */
```

The following function is called by HVE, and tells HVE what time-dependent output variables to produce. This function is user-supplied, and must be named SelectOutputTrackVars.

```
/* Function: SelectOutputTrackVars()  Version 1.0 Source Code Listing
      Copyright 1993, Engineering Dynamics Corporation
      All Rights Reserved.
*/

SHORT SelectVehicleOutputTrackVars(SHORT NumVehicles)
{
     /* Sets up the HVE vehicle output tracks.
     */

/*------------- GLOBAL VARIABLES ---------------------
VehicleOutputTrackSetup VehicleOutputTrack[MAXVEHICLES];
                                   Output track data setup structure
-------------- LOCAL VARIABLES ----------------------
     SHORT NumVehicles;            Number of vehicles
     --------------- LOCAL VARIABLES -----------------
     (none)
     /*-----------------------------------------------*/

     /* Define output variables. The sample program outputs only
        kinematic data (position, velocity and acceleration).
     */
     /* Position */
     VehicleOutputTrack[0].SMKinematics[0].Status  = NOT_EDITABLE;
     VehicleOutputTrack[0].SMKinematics[1].Status  = EDITABLE;
     VehicleOutputTrack[0].SMKinematics[2].Status  = EDITABLE;
     VehicleOutputTrack[0].SMKinematics[3].Status  = EDITABLE;
     VehicleOutputTrack[0].SMKinematics[4].Status  = EDITABLE;
     VehicleOutputTrack[0].SMKinematics[5].Status  = NOT_EDITABLE;

     /* velocity */
     VehicleOutputTrack[0].SMKinematics[13].Status = NOT_EDITABLE;

     /* acceleration */
     VehicleOutputTrack[0].SMKinematics[22].Status = NOT_EDITABLE;

     return 0;
} /* End of SelectVehicleOutputTrackVars() */
```

The following user-supplied function is called by sample.c, and sends the output data from the user's simulation to HVE. Its name is unimportant, but it must contain the function SendHveOutputectOutputTrackVars.

```
SHORT WriteOutput(void)
{
     /* Function WriteOutput() is called by any simulation to assign
        the current time-dependent simulation data to an HVE output
        track. WriteOuput returns a code from the HVE interface to
        the calling fuction in the event of user interaction or an
        error.

        Called by:        (Any simulation)
        Function Calls:   send()
     */
     /*------------- LOCAL VARIABLES ---------------------*/
     SHORT DataError = 0;           /* Data error flag        */

     /* Sprung Mass Kinematics Group (position velocity and accel)
     position */
     OutputVehicle.SMKinematics[0]    = CG_X;
     OutputVehicle.SMKinematics[1]    = CG_Y;
     OutputVehicle.SMKinematics[2]    = CG_Z;
     OutputVehicle.SMKinematics[3]    = Roll;
     OutputVehicle.SMKinematics[4]    = Pitch;
     OutputVehicle.SMKinematics[5]    = Yaw;

     /* velocity */
     OutputVehicle.SMKinematics[13]  = V;     /* forward vel*/

     /* acceleration */
     OutputVehicle.SMKinematics[22]  = a;     /* forward accel*/

     t   += dtprint;

     /* Send the output track to HVE. Return a message if something
        goes wrong.
     */
     if ( !send( (char *)&OutputVehicle,
        (long)sizeof(struct OutputVehicleData)) )
     {
       DataError = BAD_VEH_OUTPUT_TRACK_MESSAGE;
     }

     return DataError;
} /* End of WriteOutput() */
```

The following (and final) function is called by HVE, and tells HVE what messages to display by HVE during Playback (sample.c produces no messages).

```
/* Function ShutDown()   Version 1.0 Source Code Listing
      Copyright 1993, Engineering Dynamics Corporation
      All Rights Reserved.
*/

SHORT ShutDown(void)
{
   return 0;
} /* End of ShutDown() */

/* end of sample.c */
```