SAE TECHNICAL PAPER SERIES

960889

Programming FORTRAN Applications for HVE

Wesley D. Grimes
Collision Engineering Associates, Inc.

Reprinted from: Accident Reconstruction: Technology and Animation VI (SP-1150)

The Engineering Society For Advancing Mobility Land Sea Air and Space®
INTERNATIONAL

International Congress & Exposition
Detroit, Michigan
February 26-29, 1996

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

ISSN 0148-7191

Printed in USA 96-0049

960889

Programming FORTRAN Applications for HVE

Wesley D. Grimes
Collision Engineering Associates, Inc.

ABSTRACT

The Human Vehicle Environment (HVE) program, developed by Engineering Dynamics Corporation, combines the vehicle parameters, physics and graphics into a single computer system for use in analyzing motor vehicle collisions, handling issues, studying occupant motion, etc. One of the most valuable assets of the HVE program is the open architecture that allows easy access to the data and graphics capabilities from an independent computer program. Thus, virtually any program that can be recompiled on the Silicon Graphics system can be set up to utilize the HVE tools. HVE is written in two computer languages known as C and C++ (pronounced "C plus plus"), this aids in the graphics processing. Unfortunately, FORTRAN programs do not automatically interface with C or C++ programs. These programs must be modified to allow a two-way data path to and from HVE. This paper will briefly review the concepts of interfacing programs and then give specific examples of combining FORTRAN programs with the HVE environment.

INTRODUCTION

Over the past 30 years the major programming language for computer programs in the field of collision reconstruction has been FORTRAN. The majority of motor vehicle and occupant simulation programs were originally developed in the FORTRAN computer language, with many of these later ported to another language such as C.

Most of these FORTRAN programs are difficult to use because of restrictive file formats and the lack of graphics capabilities. In addition, when setting up such a program, the graphics necessary for visualizing the vehicle or occupant motion must often be re-written for each application. The HVE (Human Vehicle Environment) program, developed by Engineering Dynamics Corporation, provides these graphical interface routines as well as other utility programs that make interfacing with a 3-dimensional environment model much easier.[1,2,3]*

This paper will present basic concepts required to write and execute a FORTRAN program within the HVE environment. This discussion is meant for anyone trying to develop computer algorithms inside the HVE environment, but it will also aid an end user in understanding why HVE programs behave the way they do.

HVE PROGRAM

HVE is an interactive 3-dimensional analysis environment developed for a Silicon Graphics (SGI) computer system. The HVE environment is an open architecture much like the Windows environment on the IBM PC computers. This environment makes system functions available to the developer that allow access to sophisticated algorithms.

---

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

The HVE system has many useful analysis programs currently available from Engineering Dynamics. These programs include EDCRASH, EDSMAC, EDSVS, EDVTS, EDVSM (3-d vehicle handling program), EDVDS (3-d tractor/trailer simulation), and EDHIS (3-d occupant simulation).[4-10]

The HVE environment expands the abilities of 2-dimensional programs, such as EDSMAC, EDSVS, EDVTS, etc., by allowing the program to interact with a 3-dimensional environment. Figure 1 shows an example of an EDSMAC run with the impact taking place on a sloped roadside. These programs, as well as the HVE environment, are written in the C and C++ (pronounced "c plus plus") computer languages.

In the collision reconstruction community there are many programs available that have been written in FORTRAN. There are also many opportunities to develop more elaborate simulation models using HVE as a programming environment. The HVE environment allows analysts to concentrate on the development of the physics model and end the cycle of repeatedly developing a user interface. HVE contains the necessary groundwork for virtually any detail of physics modeling that deals with collisions involving motor vehicles, occupants, and pedestrians.[2,3]

The basic layout of the HVE system consists of three object databases; human, vehicle, and environment, along with tools to interface with physics programs. The physics programs are used in the analysis of motor vehicle, occupant, and pedestrian collisions using objects from the appropriate database.

Each database contains generic objects, such as a 4-door passenger car, as well as very specific objects, such as a 1984 Chevrolet Celebrity. The user can select the existing objects or add user-defined objects to the database as needed.

The human database contains information and specifications on several human models that can be used in developing physics programs describing how an occupant, or pedestrian, would behave in a motor vehicle collision. Figure 2 shows an example of a pedestrian impact analysis using the EDHIS program. Similarly, the vehicle database contains specifications on motor vehicles that can be used in developing physics algorithms describing motor

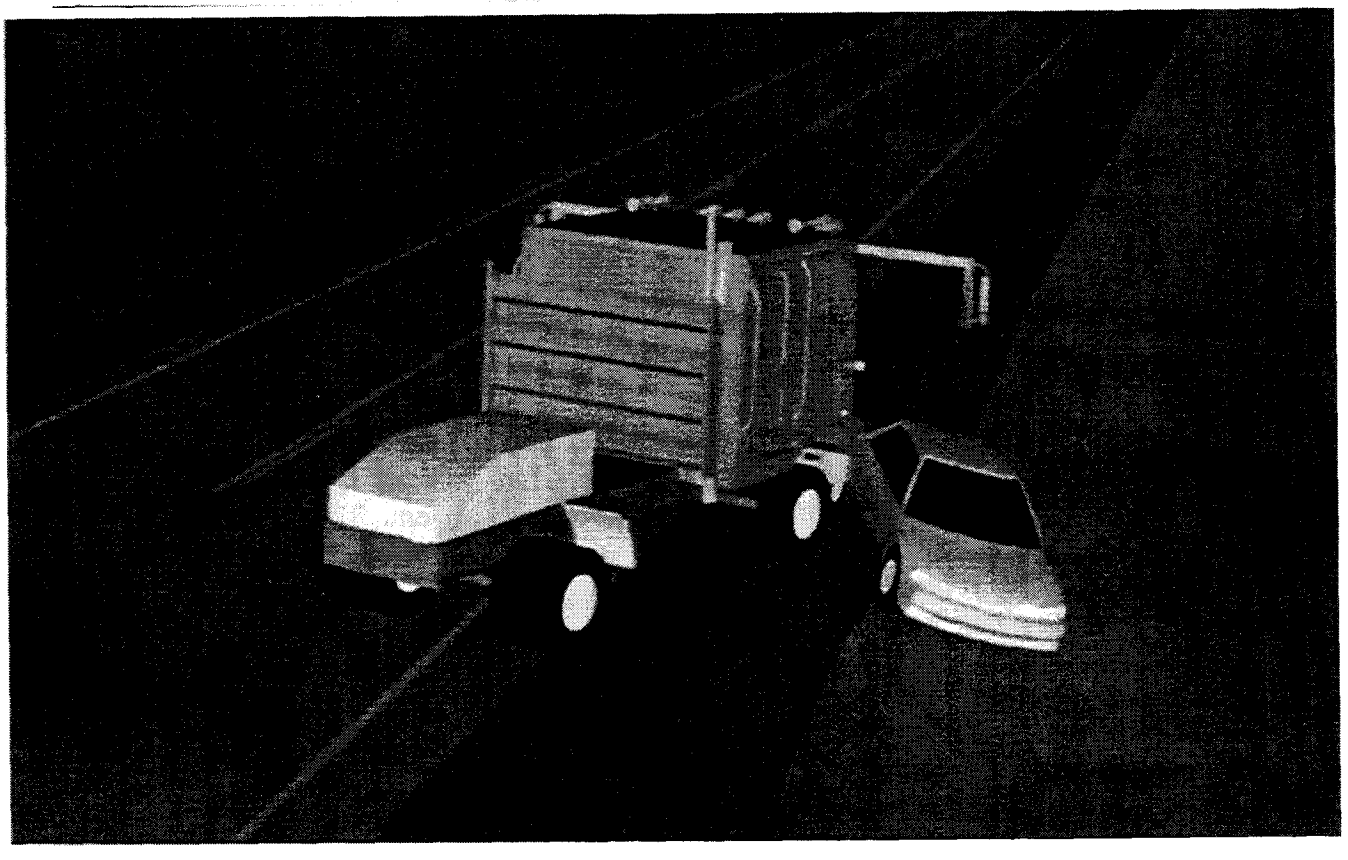

Figure 1 - EDSMAC analysis on sloped roadway shoulder.

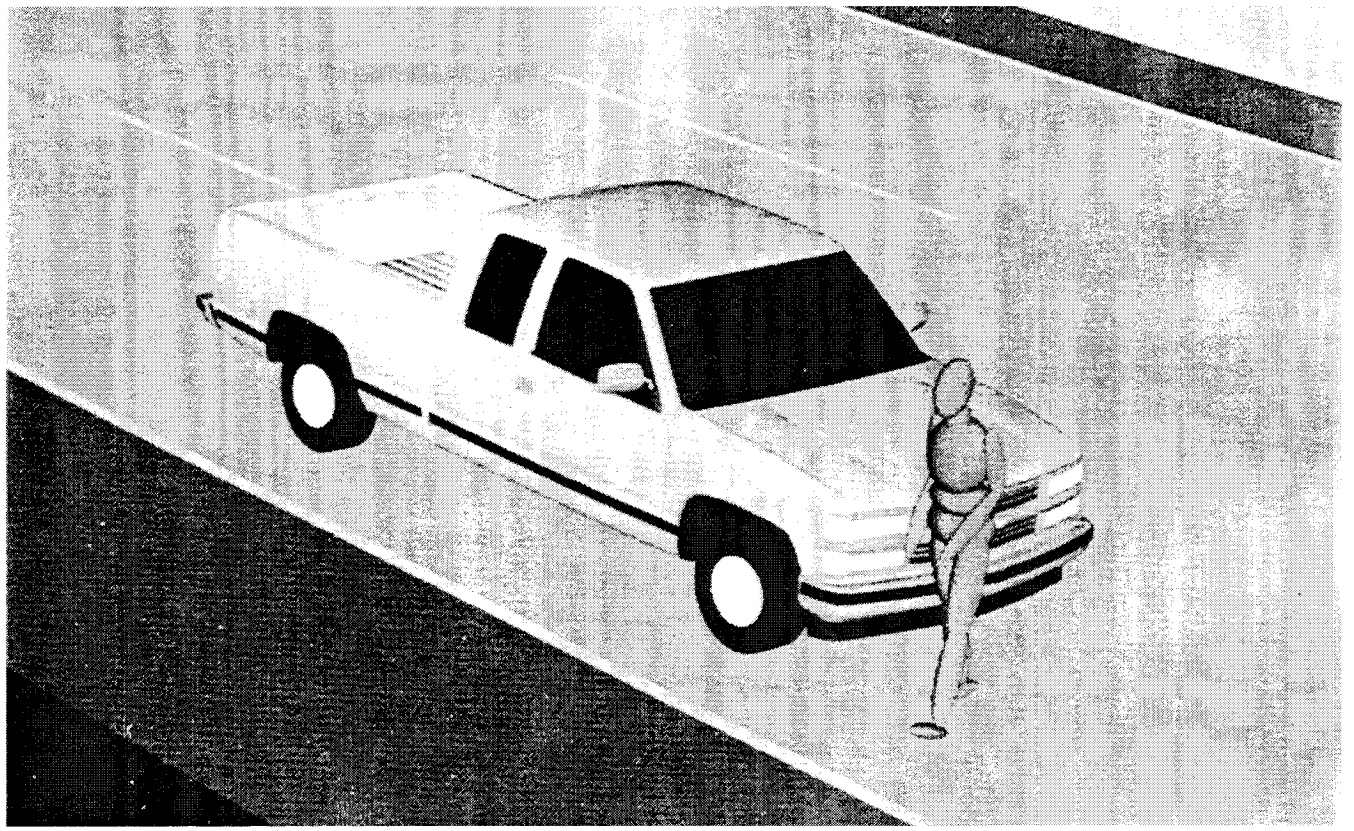

Figure 2 - Pedestrian collision analysis using HVE.

vehicle collisions, handling, overturns, etc. The environment model also contains information necessary to develop physics algorithms describing how a vehicle would interact with its surroundings. All of this data is available to an analyst or programmer through a series of function calls within the HVE environment.

FORTRAN AND C COMPARISON

SIMILARITIES - As described earlier, HVE is primarily written in the C++ computer language. The HVE physics programs currently available are written in C. There are some differences between how C and FORTRAN behave, but many similarities as well.[11] FORTRAN has COMMONs while C has STRUCTUREs; although not strictly the same, these are similar enough in concept that they can be discussed together. FORTRAN calls SUBROUTINEs and C calls FUNCTIONs; again, not strictly the same, but very similar.

ARRAY INDICES - The differences between FORTRAN and C need to also be understood. One of the main differences is that C starts all indices at zero (0), while FORTRAN starts at one (1). Thus, in FORTRAN an array x(10) contains ten elements (1 through 10), while this same x[10] in C contains ten elements (0 through 9). Note that in some enhanced versions of FORTRAN, the programmer can set what the lower array is (1 or 0), but in general most FORTRAN compilers start at one.

ARRAY STORAGE - Another difference is that FORTRAN stores two dimensional arrays in column order first, with the left-most subscript varying fastest. Thus in FORTRAN an array x(2,3) is stored as:

x(1,1), x(2,1), x(1,2), x(2,2), x(1,3), x(2,3)

In C the arrays are stored just the opposite, in row order first, with the right-most subscript varying fastest. Thus, in C this same x[2][3] array would be stored as:

x[0][0], x[0][1], x[0][2], x[1][0], x[1][1], x[1][2]

With this in mind, FORTRAN and C can pass data back and forth without problems.

In general, programs written in C and FORTRAN can pass data back and forth in FORTRAN COMMONs and C STRUCTUREs. The variables can also be passed as arguments, but it is much simpler to set up STRUCTUREs and COMMONs to accomplish this task.

VARIABLE NAMES - A peculiar thing about the FORTRAN compiler on the SGI system is that it appends an underscore (_) character to any SUBROUTINE, FUNCTION, COMMON, or variable name. All names are also converted to lower-case, unless specific compiler switches are set to allow upper- and lower-case names as unique. Thus, the FORTRAN statements;

SUBROUTINE COLL
COMMON /DATA1/ I, R

results in the names "coll_", "data1_", "i_", and "r_" being passed to the linker program. In order to interface with a C program, these variables must be referenced with the underscore, and STRUCTURES in the C program must append the COMMON name at the end of the STRUCTURE definition. Thus, to interface with this FORTRAN SUBROUTINE, the C routine would look similar to the code in Figure 3.

```
struct S { int i; float r;} data1_;
main()
{
   coll_();
   printf("%d %f \n", data1_.i, data1_.r);
}
```

Figure 3 - Code required in C to interface with FORTRAN code shown in text above.

This example simply sets up the necessary interface between the FORTRAN and C routines, then calls the COLL subroutine, and finally prints the values for the FORTRAN variables I and R.

HVE PHYSICS PROGRAM REQUIREMENTS

Every program executed in the HVE environment has very specific requirements. These requirements specify how the physics program "looks" to HVE, and this allows HVE to easily run and communicate with these separate programs. Every program developed must have at least the six C functions listed in figure 4.

ParseHveInput() - Retrieves HVE data values and places values in the physics program variables.

CalcMethodInfo() - Sets up flags and information about the physics program.

SelectVehicleOutputTrackVars() - Set up for which vehicle variables will be used in the physics algorithms and which variables will have values passed back to HVE.

SelectHumanOutputTrackVars() - Set up for which human variables will be used in the physics algorithms and which variables will have values passed back to HVE.

ExecuteCalcMethod() - Contains the actual physics algorithms, or the functions that call the physics algorithms.

ShutDown() - Contains the functions to end the program.

Figure 4 - List of six functions the developer must provide in their program for HVE.

These six C functions are called by the HveMain() program and this is what allows developer-written programs to communicate with HVE. When a program is compiled and linked with the HVE libraries, a function called "HveMain()" is linked into the executable program. When the user executes a program that has been set up to run under HVE, control is basically passed to this "HveMain()" function.

Thus, every HVE program has a "HveMain()" function in the executable code. The SGI system, running under UNIX, keeps track of all these programs so no conflicts occur even though they all have many of the same function names running at the same time.

HveMain() - As discussed above, every HVE program will have a "HveMain()" function. The developer does not write this function, it is included in the HVE libraries and is combined with the program as the program is compiled and linked. The HveMain() function is the main function for each program and this is where control is passed from HVE to the developer's program. Inside the HveMain() function, there is basically a program loop that calls the four functions:

ParseHveInput()
ExecuteCalcMethod()
CalcMethod()
ShutDown()

Note that SelectVehicleOutputTrackVars() and SelectHumanOutputTrackVars() are called by OutputTrackSetup(), which is called by HveMain(). Thus, all six required functions are either directly, or indirectly, called by HveMain().

ParseHveInput()- This user written function contains the code that takes the HVE variables and assigns them to the user's program variables. For instance, the user's program may use a variable "xpos" to represent the x-position of a vehicle in a dynamic simulation program. This data is in "EventVehicle[0].PosVel[0].XPos" inside the HVE EventVehicle structure. Recall that array indices begin with 0 in the C language. In this case, one of the assignment statements in the ParseHveInput() function would be:

xpos = EventVehicle[0].PosVel[0].XPos

Virtually all of the vehicle data is parsed out into the variables for use in the physics program inside this function. The HVE environment includes vehicle data on everything from the number of axles on a vehicle to information on driver steering or braking.

The HVE vehicle data structures are broken up into two major groups; the Vehicle and EventVehicle. The Vehicle structure contains the variables that do not change from one event to another, such as: vehicle weight, moments of inertia, suspension and tire properties, etc. The EventVehicle structure contains the variables that will change from one event to another, such as: vehicle position or orientation, vehicle speed and acceleration, etc.

The Vehicle structure defines a vehicle "sitting in the parking lot" while the EventVehicle structure specifies a vehicle unique to a certain event. All the variables used in the physics program that relate to

Code in FORTRAN routine:

```
      SUBROUTINE FTNCALC
      REAL POS, VEL, DELTAT, ENDT
      COMMON /INIT/ POS(6), VEL(6), DELTAT, ENDT
      COMMON /DATA/ N, TIME(200), EPOS(6,200)
```

Code in C routine:

```
      init_.pos[0] = EventVehicle[0].PosVel[0].XPos;
      init_.pos[1] = EventVehicle[0].PosVel[0].YPos;
      init_.pos[2] = EventVehicle[0].PosVel[0].ZPos;
      init_.pos[5] = EventVehicle[0].PosVel[0].YawOrient;

      init_.vel[0] = EventVehicle[0].PosVel[0].xVel;
      init_.vel[1] = EventVehicle[0].PosVel[0].yVel;
      init_.vel[4] = EventVehicle[0].PosVel[0].PitchVel;
      init_.vel[5] = EventVehicle[0].PosVel[0].YawVel;

      init_.deltaT = Event.Info.SimControls.dtOutput;
      init_.endT = Event.Info.SimControls.Tmax;
```

Figure 5 - Partial listing of FORTRAN and C code need for ParseHveInput() function.

```
/*-------------------------------------------------------------*/
CalcMethodHeader.NumObjects = Event.Info.NumSelectedObjects;
CalcMethodHeader.Options.IsSimulation          = TRUE;
CalcMethodHeader.Options.IsReconstruction      = FALSE;

/* 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          = TRUE;

/* 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;
CalcMethodHeader.Options.BegPerceptionPosIsUsed = TRUE;
CalcMethodHeader.Options.EndOfRotPosIsUsed     = TRUE;
CalcMethodHeader.Options.FinalPosIsUsed        = TRUE;

/* The following tell HVE to make the following edit dialogs
   available in Event mode.
*/
CalcMethodHeader.Options.DamageDataDlgIsUsed   = FALSE;
CalcMethodHeader.Options.PayloadXIsUsed        = FALSE;
CalcMethodHeader.Options.PayloadYIsUsed        = FALSE;
CalcMethodHeader.Options.PayloadZIsUsed        = FALSE;
```

Figure 6 - Partial listing of CalcMethodInfo() function written by developer.

the vehicle are parsed out in this function whether they are Vehicle specific or EventVehicle specific. There are similar data groups for occupants and pedestrians, with Human and EventHuman structures.

When writing the physics program in FORTRAN, the ParseHveInput() function will still be in the C language, but will pass the variables into the FORTRAN COMMONs. Portions of an example are shown in Figure 5, the complete routines are in Appendix A & B. There are template files for all of the required C functions available as part of the HVE Developer's Toolkit.[1] These template files make it easy to set up any program by providing all the structures, variables, etc. in a standard format that can be edited by the developer.

CalcMethodInfo() - The CalcMethodInfo() function is used to describe the calculation method being used. The HVE environment allows for reconstruction calculation methods, such as EDCRASH, or for simulation programs, such as EDSMAC, EDVTS, EDHIS, etc. This function is used to pass information back to HVE.

There are many types of information about a program that HVE requires to perform properly. This information can be broken into three categories:

1) information about the program
2) data input required or used
3) data output produced

A partial listing of a CalcMethodInfo() function is shown in Figure 6, with the entire function listed in Appendix C. This information basically lets HVE know what type of calculations will be performed and what options that HVE should allow.

The basic information that HVE needs about the program is listed below:

Type of program - Reconstruction/simulation
(human or vehicle)
Number of objects needed by the program
Number of analytical dimensions (2-d or 3-d)

The program will also need information from HVE. Some of the possible data available is shown in

Figure 6. The HVE environment will use this information to validate that the required data will be available for the program.

For instance, if the user selects a vehicle and then picks a human simulation program, the HVE program will produce an error indicating that the wrong type of object has been selected for this simulation program.

Some of the possible program input that HVE needs to have information about are listed below:

Which "target" positions are used
(Initial, POI, POR, Point of Curve, etc.)
Whether damage data is required
Whether payload data is used
Use of braking, steering, throttle tables
Vehicle acceleration pulse
(for human simulators)

The CalcMethodInfo() function also sets up the type of output that will be expected. HVE then sets up the appropriate output areas for the program to use and makes these output reports available for the user to view in the PlayBack Editor.

For example, if the selected program produces a vehicle damage diagram, then HVE will set up the necessary graphic images, etc. Some of the possible program output categories are shown in Figure 7. These options are selected with a TRUE or FALSE assignment in the CalcMethodInfo() function, as shown in the partial listing in Figure 8.

Accident History
Damage Data
Damage Profiles (graphical)
Vehicle Acceleration Pulse
Human Motion Data
Human Injury Data
Momentum Diagrams (graphical)
Program Data
Site Drawing (graphical)
Trajectory Simulation (3-d visualization)
Variable Output
Vehicle Data

Figure 7 - Some possible program output that can be identified in CalcMethodInfo().

When a program is written in FORTRAN, a simple template file can be used to pass this information to HVE. Because the FORTRAN program will not have to interface with the CalcMethodInfo() function, it is convenient to simply use a C template file such as that shown in Appendix C.

It is important to understand that indicating to HVE that one, or more, of these output reports will be available does not put the data there. All that is accomplished here is telling HVE which output reports should be made available to the user while in the Playback Editor. It is still up to the developer to place the data in these output reports... so far the program has simply promised HVE that it will be done.

```
CalcMethodHeader.Options.AccidentHistory        = TRUE;
CalcMethodHeader.Options.DamageData             = FALSE;
CalcMethodHeader.Options.DamageProfiles         = FALSE;
CalcMethodHeader.Options.HumanData              = FALSE;
CalcMethodHeader.Options.InjuryData             = FALSE;
CalcMethodHeader.Options.Messages               = TRUE;
CalcMethodHeader.Options.MomDiagramDamage       = FALSE;
CalcMethodHeader.Options.MomDiagramScene        = FALSE;
CalcMethodHeader.Options.ProgramData            = TRUE;
CalcMethodHeader.Options.Results                = FALSE;
CalcMethodHeader.Options.SiteDrawing            = FALSE;
CalcMethodHeader.Options.TrajSimulation         = TRUE;
CalcMethodHeader.Options.VariableOutput         = TRUE;
CalcMethodHeader.Options.VehicleData            = FALSE;
```

Figure 8 - Partial listing of CalcMethodInfo() showing the selection of output reports.

SelectVehicleOutputTrackVars() and SelectHumanOutputTrackVars() - Running a simulation or other physics program would not yield much insight without being able to analyze the output from these programs. The output tracks in HVE contain the time-dependent data produced by the simulation program.

In order for HVE to have this data available, the executable program must identify what data will be sent back to HVE. Part of this is accomplished in CalcMethodInfo(), as described above.

The CalcMethodInfo() function is simply used to tell HVE what types of data will be available so that appropriate output reports are available, the variables in the functions SelectVehicleOutputTrackVars() and SelectHumanOutputTrackVars() are where the specific data, that will be sent back to HVE, is identified. Also note that the only data that is affected by these two functions (Select..Vars) is the data that is available in the VariableOutput Table.

The special property of all elements in the VariableOutput table is that they are all time-dependent and can easily be plotted, printed, etc. for documentation purposes while in the Playback Editor.

There are three possible choices for each of the variables in the VariableOutput Table:

1) NOT_USED is the default and indicates that this entry will not be in the table.

2) NOT_EDITABLE indicates that the entry will be in the table but can not be changed or edited.

3) EDITABLE indicates that the entry will appear in the table and can be edited by the HVE user while viewing the entry in the Playback Editor.

Recall that in the CalcMethodInfo() function the user defines what types of data will be available to HVE. All this really does is tell HVE to allow the user to select this type of output while in the Playback Editor. One of the possible areas available is the VariableOutput Table. In order to place data in the VariableOutput table, it must be specifically identified in the variables inside the functions SelectVehicleOutputTrackVars() or SelectHumanOutputTrackVars().

An example of this is shown in the partial listings in Figure 9 and Figure 10 with the entire listings available in Appendix D and E, respectively. As before, these files can be used as templates and will

```
VehicleOutputSetup[i].VehKinematics[0]  = NOT_EDITABLE;  /* "X_CG"*/
VehicleOutputSetup[i].VehKinematics[1]  = NOT_EDITABLE;  /* "Y_CG"*/
VehicleOutputSetup[i].VehKinematics[2]  = NOT_EDITABLE;  /* "Z_CG"*/
VehicleOutputSetup[i].VehKinematics[3]  = NOT_EDITABLE;  /* "Roll"*/
VehicleOutputSetup[i].VehKinematics[4]  = NOT_EDITABLE;  /* "Pitch"*/
VehicleOutputSetup[i].VehKinematics[5]  = NOT_EDITABLE;  /* "Yaw"*/
VehicleOutputSetup[i].VehKinematics[6]  = NOT_USED;      /* "PathRad"*/
VehicleOutputSetup[i].VehKinematics[7]  = NOT_EDITABLE;  /* "Nu"*/
```

Figure 9 - Partial listing of SelectVehicleOutputTrackVars().

```
HumanOutputSetup[i].HumKinematics[j][0]  = NOT_EDITABLE;  /* "X_CG"*/
HumanOutputSetup[i].HumKinematics[j][1]  = NOT_EDITABLE;  /* "Y_CG"*/
HumanOutputSetup[i].HumKinematics[j][2]  = NOT_EDITABLE;  /* "Z_CG"*/
HumanOutputSetup[i].HumKinematics[j][3]  = NOT_EDITABLE;  /* "Roll"*/
HumanOutputSetup[i].HumKinematics[j][4]  = NOT_EDITABLE;  /* "Pitch"*/
HumanOutputSetup[i].HumKinematics[j][5]  = NOT_EDITABLE;  /* "Yaw"*/
```

Figure 10 - Partial listing of SelectHumanOutputTrackVars().

not need to be entirely re-written by a developer with each program. Figure 11 lists the Output Groups available in HVE.

For example, if you wish to include the vehicle x-position in the VariableOutput Table; you must include the following line in the CalcMethodInfo() function:

```
CalcMethodHeader.Options.VariableOutput = TRUE;
```

In addition, the following line would be required in the SelectVehicleOutputTrackVars():

```
VehicleOutputSetup[i].VehKinematics[0] = NOT_EDITABLE
```

The data would then be passed to the output report in the main physics routine, or an output subroutine. This is shown later in the paper.

ExecuteCalcMethod() - This is the function that contains the actual physics algorithm. In most cases the ExecuteCalcMethod() function contains several calls to other functions, such as InitilizeData(), ComputeVel(), etc. that actually contain the physics equations. In general, all the physics calculations take place through ExecuteCalcMethod(), whether all these equations reside in this function or other functions that are called by ExecuteCalcMethod().

The ExecuteCalcMethod() function in an HVE FORTRAN program is where execution is turned over from the C portion to the FORTRAN portion. This is also where it is convenient to send data back to HVE from the FORTRAN program.

Recall that the set up for passing data back to HVE was in the SelectVehicleOutputTrackVars() for vehicle data and the SelectHumanOutputTrackVars() for data from a human simulation program. In order to send data back to HVE from the executing program, the data is loaded into the output tracks (a data structure set up to send data back to HVE). Recall the example of sending the vehicle x-position from the executing program to HVE (discussed in the section above); in this example, the following line of code would appear in ExecuteCalcMethod() (or a function called by ExecuteCalcMethod()):

```
VehicleOutputTrac[0].VehKinematics[0] = xposn
```

Vehicle	Occupant
Kinematics	Kinematics
Kinetics	Joints
Tires	Contacts
Wheels	Belts
Connections	Airbags
Drive Train	
Driver	
Contacts	
Belts	
Airbags	

Figure 11 - List of Output Groups in HVE.

Once all the output tracks are "loaded" with appropriate data, a call is made to another HVE function: SendHveOutput(), which actually sends the data in each of the output tracks back to HVE.

Usually a developer will perform all the calculations in a FORTRAN subroutine and then use another C function (called from ExecuteCalcMethod()), such as WriteOutput(), to send the data back to HVE. The developer can either store this time-dependent data in arrays and process the data all at once at the end, or set up the program so that the appropriate data is sent back to HVE with each time iteration. The second option is preferable, in that it keeps the HVE rendered image in step with where the physics program is in time. An example of using data arrays is included in Appendix A, F, and G.

OTHER OUTPUT REPORTS - There are other functions and methods available to place data in the other output areas identified in the function CalcMethodInfo(). These other output reports consist of both graphical and numerical/text reports. There are several utility functions available from HVE that allow the developer to quickly produce output reports.

Some of the code to produce text output reports is shown in Figure 12. Note that there are no strict guidelines for what specific data goes to which report, but the developer is encouraged to place the data in the most appropriate output report.

```
      /* ACCIDENT HISTORY*/

      if ((retVal = hve_open_page(ORAccidentHistory, 0)) < 0) {
          DB(fprintf (stderr,"Could not open report ORAccidentHistory\n"));
          /* Calling HVEShutDown here ensures that the error message */
          /* will get thru and hve will not die */
          HveShutdown(retVal);
          return (retVal);
      }

      hve_center_name(MaxVeh, nameBuff, "ACCIDENT HISTORY");
      hve_string("\n");
      hve_string(nameBuff);
      hve_string("\n\n");
      hve_string ("-------------------------------------------------\n");

      himp[i][j]/17.6, V_imp[i][j]/17.6, R_imp[i]/RAD);
          hve_string(buffer);
      }
      hve_string ("\nSpeed Change, Linear Momentum\n");
      hve_string     ("           Total          Fwd          Lat       PDOF\n");
      for (I=0;i<MaxVeh;i++) {
            sprintf (buffer,"Veh %d    %7.2f mph  %7.2f mph  %7.2f mph  %7.2f
deg\n",I+1, Delvr[i][j]/17.6, Delvx[i][j]/17.6, Delvy[i][j]/17.6, Pdof[i][j]/RAD);
          hve_string(buffer);
      }
      hve_string ("\nImpact Speed, Damage\n");
      j = 2;
      hve_string     ("           Total          Fwd          Lat      Angular\n");
      for (I=0;i<MaxVeh;i++) {
            sprintf (buffer,"Veh %d    %7.2f mph  %7.2f mph  %7.2f mph  %7.2f
deg/sec\n",I+1, Vtot_imp[i][j]/17.6, U_imp[i][j]/17.6, V_imp[i][j]/17.6, R_imp[i]/RAD);
          hve_string(buffer);
      }retVal);
      }

.
.
.
      /* MESSAGES */
      if ((retVal = hve_open_page(ORMessages, 0)) < 0) {
          DB(fprintf (stderr,"Could not open report ORMessages\n"));
          fprintf (stderr,"Could not open report ORMessages\n");
          HveShutdown(retVal);
          return (retVal);
      }
      hve_center_name(MaxVeh, nameBuff, "MESSAGES");
      hve_string("\n");
      hve_string(nameBuff);
      hve_string("\n\h; VehNum++)
.
.
.
      hve_center_name(MaxVeh, nameBuff, "PROGRAM DATA");
      hve_string("\n");
      hve_string(nameBuff);
      hve_string("\n\n");
      hve_close_page();
```

Figure 12 - Example of code for additional text output reports.

For example, a warning message concerning an error encountered during program execution could be located in any of the text reports, but an end user would probably expect these messages in either the Messages or Results report.

The graphical output reports, listed in Figure 7, are extremely easy to set up for the developer. The program simply needs to place the data in the appropriate variable names and then HVE canproduce the graphical images. The developer does not need to concern themselves with any type of graphical interface programming. Some examples of coding for the graphical output reports is shown in Figure 13.

ShutDown() - This is the final developer-written function. The ShutDown() function is used to write the final output data, any diagnostic messages, and any other program data that is not time-dependant to the output reports. This function then closes down the physics program and returns control to HveMain(). An example of the ShutDown() function that can be used as a template is included in the paper as Appendix H.

GENERAL DEVELOPER ISSUES

When a user selects an object, for example a car, and then selects a physics application, such as EDSVS, HVE calls the physics program and performs a "verify". This "verify" is used to ensure that the object can be used with the selected physics program, that the program does not require additional objects, and in general determines that this program should be able to run.

For this to occur, HveMain actually runs ParseHveInput() and CalcMethodInfo() to be as certain as possible that the program will operate without a major error. Developers then have an opportunity to check what types of data are being presented to their programs and refuse to execute the program if it is not appropriate. For example, when a user selects a 1988 Chevrolet Camaro from the vehicle database and then selects EDHIS (a human dynamic simulator), the program should not be allowed to proceed past the "verify", because there is no human included in the event setup.

The other item to be sure and handle as a developer is the fact that both ParseHveInput() and CalcMethodInfo() will run twice, back to back when the program is executed. These functions will run once during the "verify" and then a second time when the program is actually started from the Event Editor. Good programming practice is to initialize all variables prior to using them, this is a good reason to do it.

It is also important to realize that during the "verify", there are no initial positions, orientations, velocities, etc. The user has not had the opportunity to enter these data yet because the "verify" takes place immediately after pressing the "okay" button on the Event Information dialog box.

After the program is compiled and linked with the HVE Developer's Toolkit libraries, the executable program is simply placed in the directory "hve/supportFiles/calcMethods/" and it will appear on the list of physics programs available in the Event Information dialog.

SUMMARY

The HVE environment is an extremely powerful tool for use in analyzing motor vehicle collisions. There are currently many tools available from Engineering Dynamics Corporation to handle a variety of simulation requirements, but there are also many other programs in existence that could greatly benefit from running under the HVE environment. The existing FORTRAN programs can be slightly modified to interface with the HVE program. Using the HVE programming environment, developers can concentrate on the physics algorithms and not be required to develop a user interface.

There is no practical way around using C when writing a FORTRAN program for the HVE environment at this time. However, the large majority of C program code required is available in template files that can easily be modified for each individual program.

It is theoretically possible to interface with HVE entirely in FORTRAN, but the existing utilities

```
/* Site Drawing
*/
SiteDrawing.NumVehicles = MaxVeh;
  for (VehNum=0; VehNum<MaxVeh; VehNum++)
{
   /* Assign vehicle IDs
   */
   SiteDrawing.Results[VehNum].Id = Vehicle[VehNum].Id;
   for (i=0; i<MAXPOSITIONS; i++)  /* for each possible position...*/
   {
      if (SiteDrawing.Results[Vehing.Results[VehNum].PosVel[i].TotalVel
                                          = Vtot_imp[VehNum][Colisn_type];

            SiteDrawing.Results[VehNum].PosVel[i].YawVel    = R_imp[VehNum];
            SiteDrawing.Results[VehNum].PosVel[i].SlipAngle = Beta_imp[VehNum];
               /* break; */


          /* case 4 : Separation position (in, rad) */
               SiteDrawing.Results[VehNum].PosVel[i+1].xVel    = U_sep[VehNum];
               SiteDrawing.Results[VehNum].PosVel[i+1].yVel    = V_sep[VehNum];
               SiteDrawing.Results[VehNum].PosVel[i+1].TotalVel= Sep_vel[VehNum];
               SiteDrawing.Results[VehNum].PosVel[i+1].YawVel  = R_sep[VehNum];

               break;

          case 5 : /* Point-on-curve position (in, rad) */

                SiteDrawing.Results[VehNum].PosVel[i].PosVel[i].YawVel    = 0.0;

               break;

          default :
                  break;
       } /* End of switch */
      }
   }
  } /* end of for VehNum */
```

Figure 13 - Example of code to produce other graphical output reports.

already written in C simply outweigh any benefit from using only FORTRAN. As third-party programs are developed, it might be possible to develop a series of HVE utilities written in FORTRAN, but it would be an enormous undertaking to produce all the utilities that currently exist.

TRADEMARKS

REFERENCES

1. Day, T.D., "An Overview of the HVE Developer's Toolkit", SAE Paper No. 940923, Society of Automotive Engineers, Warrendale, PA, 1994.

2. Day, T.D., "An Overview of the HVE Vehicle Model", SAE Paper No. 950308, Society of Automotive Engineers, Warrendale, PA, 1995.

3. Day, T.D., "An Overview of the HVE Human Model", SAE Paper No. 950659, Society of Automotive Engineers, Warrendale, PA, 1995.

4. EDCRASH/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR, Under development.

5. EDSMAC/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR,Under development.

6. EDSVS/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR, Under development.

7. EDVTS/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR,Under development.

8. EDVSM/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR, Under development.

9. EDVDS/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR,Under development.

10. EDHIS/HVE Progam Manual, Engineering Dynamics Corporation, Beaverton, OR,Under development.

11. FORTRAN 77 Programmer's Guide, Silicon Graphics, Inc., Mt View, CA.

Appendix A
FORTRAN program setup for interfacing with HVE

```
      SUBROUTINE FTNCALC
C*************************************************************************
C     Written by Collision Engineering Assoiates, Inc - Daniel J. Kuhn
C                                                     - Wesley D. Grimes
C          Last Update: 11/8/95
C
C     Called By: ComputeData() - A C function
C
C     This routine is the Fortran Physics package sample that can be
C     interfaced with HVE and it's C libraries.
C
C     This sample reads HVE's initial settings for the Initial Condition
C     target position.  These settings are:
C              Initial Positions X,Y,Z, Roll, Pitch, Yaw
C              Initial Velocity  u, v, w and Roll, Pitch and Yaw
C     Then it applies the velocities using the time step until the end time
C     is reached. All data is computed in one pass through this routine.  The
C     C function ComputeData() performs the output track loading for all the
C     data.
C*************************************************************************
C       These are the variable definitions and common blocks that get
C       shared with the C routines.  Data byte sizes that match are crucial
C       when interfacing
C
C       six element arrays provide positions and velocity for components
C            X, Y, Z, Roll, Pitch, and Yaw respectively
C
      REAL POS, VEL, DELTAT, ENDT
      COMMON /INIT/ POS(6), VEL(6), DELTAT, ENDT
      COMMON /DATA/ N, TIME(200), EPOS(6,200)
C
C       the equivelant C structures are as follows...
C
C                 struct initialVehicleData
C                 {
C               float pos[6];
C               float vel[6];
C               float deltaT, endT;
C            }init_;
C
C
C            struct eventData
C            {
C              int n;
C              float tim[200];
C              float pos[200][6];
C            }data_;
C
C-------------------------------------------------------------------------
C
C       ---- First position is the same as the intitial condition --
      TIME(1) = 0
      EPOS(1,1) = POS(1)
      EPOS(2,1) = POS(2)
      EPOS(3,1) = POS(3)
      EPOS(4,1) = POS(4)
      EPOS(5,1) = POS(5)
      EPOS(6,1) = POS(6)
```

---------- LISTING CONTINUES ON FOLLOWING PAGE ----------

```
C       -- I = counter for number of points in array --
        I = 2
C       --- Loop for the reast of the time computing next position ---
C       ----  This is where any REAL physics model calculations would go ...
        DO 100 T = DELTAT, ENDT, DELTAT
          TIME(I) = T
          DO 150 IPOS = 1, 6
             EPOS(IPOS,I) = EPOS(IPOS,I-1) + VEL(IPOS)*DELTAT
 150       CONTINUE
          I = I + 1
 100    CONTINUE

C      ----- Load the number of array elements into common for use in C ----
        N = I - 1
        RETURN
        END
```

Appendix B
EXAMPLE OF ParseHveInput()

```
INT ParseHveInput(void)
{
      /* Assigns input data used by FORTRAN Example

         Called by:           HveMain()
         Calls:               (none)
      */

/* Structures defined in header files */

      INT DataError = 0;          /* return flag                          */
      numVehicles = 1;  /* hard code objects*/
      numHumans = 0;

/* --------------------  get info on objects setup in HVE -----------
  ---------------------       for this program ------------------------*/

/* put error checking of file definition vs HVE definition here */

      if (Event.Info.NumSelectedVehicles  != numVehicles)
      {
         DataError = ERROR_NUM_OBJECTS_NOT_COMPATIBLE;
         printf("Data file contains %i vehicles and HVE has %i
defined.\n",numVehicles,Event.Info.NumSelectedVehicles);
      }
      else if (Event.Info.NumSelectedHumans != numHumans)
      {
         DataError = ERROR_NUM_OBJECTS_NOT_COMPATIBLE;
         printf("Data file contains %i humans and HVE has %i
defined.\n",numHumans,Event.Info.NumSelectedHumans);
        }

      if (DataError != 0) return DataError;

      /* Assign interface variables.
        assume initial position for 1st vehicle velocity specified */

      init_.pos[0] = EventVehicle[0].PosVel[0].XPos;
      init_.pos[1] = EventVehicle[0].PosVel[0].YPos;
      init_.pos[2] = EventVehicle[0].PosVel[0].ZPos;
      init_.pos[3] = EventVehicle[0].PosVel[0].RollOrient;
      init_.pos[4] = EventVehicle[0].PosVel[0].PitchOrient;
      init_.pos[5] = EventVehicle[0].PosVel[0].YawOrient;

      init_.vel[0] = EventVehicle[0].PosVel[0].xVel;
      init_.vel[1] = EventVehicle[0].PosVel[0].yVel;
      init_.vel[2] = EventVehicle[0].PosVel[0].zVel;
      init_.vel[3] = EventVehicle[0].PosVel[0].RollVel;
      init_.vel[4] = EventVehicle[0].PosVel[0].PitchVel;
      init_.vel[5] = EventVehicle[0].PosVel[0].YawVel;

      init_.deltaT = Event.Info.SimControls.dtOutput;
      init_.endT = Event.Info.SimControls.Tmax;

      return DataError;

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

Appendix C
EXAMPLE OF 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 i,j,k;            /* Local indices                        */
      /*--------------------------------------------------------------*/
      CalcMethodHeader.NumObjects = Event.Info.NumSelectedObjects;
      CalcMethodHeader.Options.IsSimulation          = TRUE;
      CalcMethodHeader.Options.IsReconstruction      = FALSE;

      /* 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          = TRUE;

      /* 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;
      CalcMethodHeader.Options.BegPerceptionPosIsUsed = TRUE;
      CalcMethodHeader.Options.BegBrakingPosIsUsed   = TRUE;
      CalcMethodHeader.Options.ImpactPosIsUsed       = TRUE;
      CalcMethodHeader.Options.SeparationPosIsUsed   = TRUE;
      CalcMethodHeader.Options.PointOnCurvePosIsUsed = TRUE;
      CalcMethodHeader.Options.EndOfRotPosIsUsed     = TRUE;
      CalcMethodHeader.Options.FinalPosIsUsed        = TRUE;

      /* The following tell HVE to make the following edit dialogs
         available in Event mode.
      */
      CalcMethodHeader.Options.DamageDataDlgIsUsed   = FALSE;
      CalcMethodHeader.Options.PayloadXIsUsed        = FALSE;
      CalcMethodHeader.Options.PayloadYIsUsed        = FALSE;
      CalcMethodHeader.Options.PayloadZIsUsed        = FALSE;
      CalcMethodHeader.Options.PayloadRollIsUsed     = FALSE;
      CalcMethodHeader.Options.PayloadPitchIsUsed    = FALSE;
      CalcMethodHeader.Options.PayloadYawIsUsed      = FALSE;
      CalcMethodHeader.Options.ThrottleWOTIsUsed     = FALSE;
      CalcMethodHeader.Options.ThrottleTractiveEffortIsUsed = FALSE;
      CalcMethodHeader.Options.ThrottleFrictionIsUsed = FALSE;
      CalcMethodHeader.Options.BrakesPedalForceIsUsed = FALSE;
      CalcMethodHeader.Options.BrakesWheelForceIsUsed = FALSE;
      CalcMethodHeader.Options.BrakesFrictionIsUsed  = FALSE;
      CalcMethodHeader.Options.WheelDataDlgIsUsed    = FALSE;
      CalcMethodHeader.Options.GearTableDlgIsUsed    = FALSE;
      CalcMethodHeader.Options.SteerAtStrgWheelIsUsed = FALSE;
      CalcMethodHeader.Options.SteerAtTiresIsUsed    = FALSE;
      CalcMethodHeader.Options.CollisionPulseDlgIsUsed= FALSE;
      CalcMethodHeader.Options.ProducesCollisionPulse = FALSE;
      CalcMethodHeader.Options.BeltRestraintsIsUsed  = FALSE;
      CalcMethodHeader.Options.AirbagRestraintsIsUsed = FALSE;
      CalcMethodHeader.Options.ContactsDlgIsUsed     = FALSE;
```

---------- LISTING CONTINUES ON FOLLOWING PAGE ----------

----- CONTINUED LISTING OF CalcMethodInfo() IN APPENDIX C ------

```
      /* The following tell HVE to make the following output dialogs
         available in Playback mode.
      */
      CalcMethodHeader.Options.AccidentHistory        = TRUE;
      CalcMethodHeader.Options.DamageData             = FALSE;
      CalcMethodHeader.Options.DamageProfiles         = FALSE;
      CalcMethodHeader.Options.HumanData              = FALSE;
      CalcMethodHeader.Options.InjuryData             = FALSE;
      CalcMethodHeader.Options.Messages               = TRUE;
      CalcMethodHeader.Options.MomDiagramDamage       = FALSE;
      CalcMethodHeader.Options.MomDiagramScene        = FALSE;
      CalcMethodHeader.Options.ProgramData            = TRUE;
      CalcMethodHeader.Options.Results                = FALSE;
      CalcMethodHeader.Options.SiteDrawing            = FALSE;
      CalcMethodHeader.Options.TrajSimulation         = TRUE;
      CalcMethodHeader.Options.VariableOutput         = TRUE;
      CalcMethodHeader.Options.VehicleData            = FALSE;

        /* Set up objectIDs for NumObjects (includes main segments only!)
      */
        j = 0;
      for (i=0; i<CalcMethodHeader.NumObjects; i++)
      {
         CalcMethodHeader.Object[i].ObjectID
                              = Event.Info.SelectedObjectIDs[j++];

         for (k=0; k<NumSubSegs[i]; k++)
         {
            /* For each main segment, which subsegment are
             attached to it?
            */
            CalcMethodHeader.Object[i].WhichIDs[k]
                                    = Event.Info.SelectedObjectIDs[j++];
         }

         /* FOR HUMAN SIMULATORS ONLY! (otherwise, comment the following out)
            What is coordinate system of main segment? All segments are
            initialized (-1L) relative to earth. However, human occupants
            are positioned relative to vehicle. (Following logic assumes
            only one human and vehicle.)

           if (IsHuman(CalcMethodHeader.Object[i].ObjectID))
         {
            if (IsOccupant)
            {
             CalcMethodHeader.Object[i].RelativeCoordSystemID = Vehicle[0].Id;
            }
           }
         */

      } /* end of set-up object IDs */
      return 0;

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

Appendix D
EXAMPLE OF SelectVehicleOutputTrackVars()

```
/*  Function: SelectVehicleOutputTrackVars()     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 ---------------------*/
      SHORT i,j,k,l;         /* Vehicle, axle, side & dual tire indices*/
      /*-------------------------------------------------------------*/
      /* Define output variables for each vehicle.
      */
      for (i=0; i<NumVehicles; i++)
      {
         /* Set up correct ID */
         VehicleOutputSetup[i].Id = Vehicle[i].Id;

         /* Position */
         VehicleOutputSetup[i].VehKinematics[0]  = NOT_EDITABLE;  /* "X_CG"*/
         VehicleOutputSetup[i].VehKinematics[1]  = NOT_EDITABLE;  /* "Y_CG"*/
         VehicleOutputSetup[i].VehKinematics[2]  = NOT_EDITABLE;  /* "Z_CG"*/
         VehicleOutputSetup[i].VehKinematics[3]  = NOT_EDITABLE;  /* "Roll"*/
         VehicleOutputSetup[i].VehKinematics[4]  = NOT_EDITABLE;  /* "Pitch"*/
         VehicleOutputSetup[i].VehKinematics[5]  = NOT_EDITABLE;  /* "Yaw"*/
         VehicleOutputSetup[i].VehKinematics[6]  = NOT_USED;      /* "PathRad"*/
         VehicleOutputSetup[i].VehKinematics[7]  = NOT_EDITABLE;  /* "Nu"*/

         /* velocity */
         VehicleOutputSetup[i].VehKinematics[8]  = NOT_EDITABLE;  /* "Vtotal"*/
         VehicleOutputSetup[i].VehKinematics[9]  = NOT_EDITABLE;  /* "Beta"*/
         VehicleOutputSetup[i].VehKinematics[10] = NOT_EDITABLE;  /* "uvel" (long)*/
         VehicleOutputSetup[i].VehKinematics[11] = NOT_EDITABLE;  /* "vvel" (side)*/
         VehicleOutputSetup[i].VehKinematics[12] = NOT_EDITABLE;  /* "wvel" (normal)*/
         VehicleOutputSetup[i].VehKinematics[13] = NOT_USED;      /* "fwdvel"*/
         VehicleOutputSetup[i].VehKinematics[14] = NOT_USED;      /* "latvel"*/
         VehicleOutputSetup[i].VehKinematics[15] = NOT_EDITABLE;  /* "rollvel"*/
         VehicleOutputSetup[i].VehKinematics[16] = NOT_EDITABLE;  /* "pitchvel"*/
         VehicleOutputSetup[i].VehKinematics[17] = NOT_EDITABLE;  /* "yawvel"*/

         /* acceleration */
         VehicleOutputSetup[i].VehKinematics[18] = NOT_EDITABLE;  /* "Atotal"*/
         VehicleOutputSetup[i].VehKinematics[19] = NOT_EDITABLE;  /* "udot"*/
         VehicleOutputSetup[i].VehKinematics[20] = NOT_EDITABLE;  /* "vdot"*/
         VehicleOutputSetup[i].VehKinematics[21] = NOT_EDITABLE;  /* "wdot"*/
         VehicleOutputSetup[i].VehKinematics[22] = NOT_USED;      /* "fwdacc"*/
         VehicleOutputSetup[i].VehKinematics[23] = NOT_USED;      /* "latacc"*/
         VehicleOutputSetup[i].VehKinematics[24] = NOT_USED;      /* "tangacc"*/
         VehicleOutputSetup[i].VehKinematics[25] = NOT_USED;      /* "centacc"*/
         VehicleOutputSetup[i].VehKinematics[26] = NOT_EDITABLE;  /* "pdot"*/
         VehicleOutputSetup[i].VehKinematics[27] = NOT_EDITABLE;  /* "qdot"*/
         VehicleOutputSetup[i].VehKinematics[28] = NOT_EDITABLE;  /* "rdot"*/
         VehicleOutputSetup[i].VehKinematics[29] = NOT_USED;      /* "KTtrans"*/
```

---------- LISTING CONTINUES ON FOLLOWING PAGE ----------

```
        VehicleOutputSetup[i].VehKinematics[30] = NOT_USED;      /* "KTrotn"*/
        VehicleOutputSetup[i].VehKinematics[31] = NOT_USED;      /* "KTtotal"*/

    /* The following output groups are wheel-dependent */

    /* 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'.
    */
    for (j=0; j<Vehicle[0].NumAxles; j++) {
       for (k=0; k<2; k++)  {   /* 0 left & right sides */
        for (l=0; l<1+(SHORT)Vehicle[i].Wheel[j][j].Tire.IsDual; l++)  {
           /* l = 2 for dual tires */
             VehicleOutputSetup[i].TireData[j][k][l][0]  = NOT_USED; /* x  */
             VehicleOutputSetup[i].TireData[j][k][l][1]  = NOT_USED; /* y  */
             VehicleOutputSetup[i].TireData[j][k][l][2]  = NOT_USED; /* z  */
             VehicleOutputSetup[i].TireData[j][k][l][3]  = NOT_USED; /* X  */
             VehicleOutputSetup[i].TireData[j][k][l][4]  = NOT_USED; /* Y  */
           VehicleOutputSetup[i].TireData[j][k][l][5]  = NOT_USED; /* Z  */
             VehicleOutputSetup[i].TireData[j][k][l][6]  = NOT_USED; /* Fx'*/
             VehicleOutputSetup[i].TireData[j][k][l][7]  = NOT_USED; /* Fy'*/
             VehicleOutputSetup[i].TireData[j][k][l][8]  = NOT_USED; /* Fz'*/
             VehicleOutputSetup[i].TireData[j][k][l][9]  = NOT_USED; /* Mx'*/
             VehicleOutputSetup[i].TireData[j][k][l][10] = NOT_USED; /* My'*/
             VehicleOutputSetup[i].TireData[j][k][l][11] = NOT_USED; /* Mz'*/
             VehicleOutputSetup[i].TireData[j][k][l][12] = NOT_USED;
                                                    /* loaded tire rad*/
           VehicleOutputSetup[i].TireData[j][k][l][13] = NOT_USED;
                                                    /* long tire slip */
           VehicleOutputSetup[i].TireData[j][k][l][14] = NOT_USED;
                                                    /* slip ang,alpha */
             VehicleOutputSetup[i].TireData[j][k][l][15] = NOT_USED;
                                                      /* skid flg*/
             VehicleOutputSetup[i].TireData[j][k][l][16] = NOT_USED;
                                                      /* scuf flg*/
          } /* next tire

        /* Wheel Group (kinematics & forces) relative to
           vehicle-fixed coordinate system
        */
        /* position of wheel center */
          VehicleOutputSetup[i].WheelData[j][k][0] = NOT_EDITABLE;/*x */
          VehicleOutputSetup[i].WheelData[j][k][1] = NOT_EDITABLE;/*y */
          VehicleOutputSetup[i].WheelData[j][k][2] = NOT_EDITABLE;/*z */
          VehicleOutputSetup[i].WheelData[j][k][3] = NOT_EDITABLE;
                                                      /* cmbr ang*/
          VehicleOutputSetup[i].WheelData[j][k][4] = NOT_EDITABLE;
                                                      /* spin ang*/
        VehicleOutputSetup[i].WheelData[j][k][5] = NOT_EDITABLE;
                                                   /* delta, steer ang*/

        /* velocity */
          VehicleOutputSetup[i].WheelData[j][k][6] = NOT_USED; /* xdot*/
          VehicleOutputSetup[i].WheelData[j][k][7] = NOT_USED;
                                                     /* gamma-dot */
          VehicleOutputSetup[i].WheelData[j][k][8] = NOT_USED;
                                  /* angular veloc about spin axis */

        /* acceleration */
          VehicleOutputSetup[i].WheelData[j][k][9]  = NOT_USED;/*xddot*/
                                  /* normal accel of wheel ctr */
```

---------- LISTING CONTINUES ON FOLLOWING PAGE ----------

```
            VehicleOutputSetup[i].WheelData[j][k][10] = NOT_USED;
                                                    /* gamma-ddot */
            VehicleOutputSetup[i].WheelData[j][k][11] = NOT_USED;
                                    /* angular accel about spin axis */


          /*forces */
            VehicleOutputSetup[i].WheelData[j][k][12] = NOT_USED; /* Fx */
            VehicleOutputSetup[i].WheelData[j][k][13] = NOT_USED; /* Fy */
            VehicleOutputSetup[i].WheelData[j][k][14] = NOT_USED; /* Fz */


          /* suspension deflection and forces */
            VehicleOutputSetup[i].WheelData[j][k][15] = NOT_USED;
                                  /* wheel deflection from equilibrium */
            VehicleOutputSetup[i].WheelData[j][k][16] = NOT_USED;
                                               /* wheel deflection rate */

          /* spring force (at wheel) */
          VehicleOutputSetup[i].WheelData[j][k][17] = NOT_USED; /* Fz */

          /* damping force (at wheel) */
          VehicleOutputSetup[i].WheelData[j][k][18] = NOT_USED; /* Fz */

          /* antipitch force (at wheel) */
            VehicleOutputSetup[i].WheelData[j][k][19] = NOT_USED; /* Fz */

            /* Results of driver controls (attempted throttle and
             brake torque, and steer angle at each wheel)
          */
            /* Drive torque about spin axis */
          VehicleOutputSetup[i].WheelData[j][k][20] = NOT_USED;

              /* Brake torque about spin axis */
          VehicleOutputSetup[i].WheelData[j][k][21] = NOT_USED;

              /* Brake pressure at wheel */
          VehicleOutputSetup[i].WheelData[j][k][22] = NOT_USED;

              /* Brake temperature at wheel */
          VehicleOutputSetup[i].WheelData[j][k][23] = NOT_USED;

              /* steer angle (delta) at wheel */
          /* See wheel position, variable No. 5 */


        } /* next side */
      } /* next axle */


      /* 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 */
      VehicleOutputSetup[i].Connection[0] = NOT_USED; /* roll artic    */
      VehicleOutputSetup[i].Connection[1] = NOT_USED; /* pitch artic   */
      VehicleOutputSetup[i].Connection[2] = NOT_USED; /* yaw artic     */
```

---------- LISTING CONTINUES ON FOLLOWING PAGE ----------

```
        /* velocity */
        VehicleOutputSetup[i].Connection[3] = NOT_USED; /* roll artic vel */
        VehicleOutputSetup[i].Connection[4] = NOT_USED; /* pitch artic vel*/
        VehicleOutputSetup[i].Connection[5] = NOT_USED; /* yaw artic vel  */

        /* acceleration */
        VehicleOutputSetup[i].Connection[6] = NOT_USED; /* roll art accel */
        VehicleOutputSetup[i].Connection[7] = NOT_USED; /* pitch art accel*/
        VehicleOutputSetup[i].Connection[8] = NOT_USED; /* yaw art accel  */

        /* connection forces and moments */
        VehicleOutputSetup[i].Connection[9]  = NOT_USED;/* Fx connection  */
        VehicleOutputSetup[i].Connection[10] = NOT_USED;/* Fy connection  */
        VehicleOutputSetup[i].Connection[11] = NOT_USED;/* Fz connection  */
        VehicleOutputSetup[i].Connection[12] = NOT_USED;/* Mx connection  */
        VehicleOutputSetup[i].Connection[13] = NOT_USED;/* My connection  */
        VehicleOutputSetup[i].Connection[14] = NOT_USED;/* Mz connection  */

        /* Drivetrain Group (engine, transmission, differential) */
        /* engine */
        VehicleOutputSetup[i].Drivetrain[0]  = NOT_USED;/* engine speed   */
        VehicleOutputSetup[i].Drivetrain[1]  = NOT_USED;/* engine power   */
        VehicleOutputSetup[i].Drivetrain[2]  = NOT_USED;/* engine torque  */

        /* transmission and differential gear ratios */
        VehicleOutputSetup[i].Drivetrain[3]  = NOT_USED;/*trans gear ratio*/
        VehicleOutputSetup[i].Drivetrain[4]  = NOT_USED;/*diff gear ratio */

        /* Driver Controls Group (throttle, brake, gear, steering) */
        VehicleOutputSetup[i].Driver[0]      = NOT_USED;/* throt position */
        VehicleOutputSetup[i].Driver[1]      = NOT_USED;/* brk pedal force*/
        VehicleOutputSetup[i].Driver[2]      = NOT_USED;/* brk sys press  */
        VehicleOutputSetup[i].Driver[3]      = NOT_USED;/* trans gear no. */
        VehicleOutputSetup[i].Driver[4]      = NOT_USED;/* diff gear no.  */
        VehicleOutputSetup[i].Driver[5]      = NOT_USED;/* steer wheel ang*/

      } /* End for i */
     printf("Exiting AtbLink routine - SelectVehicleOutputTrackVars()\n");

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

Appendix E
EXAMPLE OF SelectHumanOutputTrackVars()

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

INT SelectHumanOutputTrackVars(SHORT NumHumans)
{
      /* Sets up the HVE vehicle output tracks.

                    Called By:    OutputTrackSetup()    (hvemain.c)
                   Calls:        (none)

      */

      /*------------------------ GLOBAL VARIABLES ---------------------
      HumanOutputTrackSetup HumanOutputTrack[MAXVEHICLES];
                             Output track data setup structure
      ------------------------- LOCAL VARIABLES ----------------------
      SHORT NumHumans;          Number of humans
      ------------------------- LOCAL VARIABLES ----------------------*/
      INT DataError = 0;       /* Error flag returned to caller        */
      /*--------------------------------------------------------------*/
      INT i,j;

      /* Define output variables for each Human.
      */
      return 0;
      for (i=0; i<NumHumans; i++)
      {

         /* Set up correct ID */
         HumanOutputSetup[i].Id = Human[i].Id;

         /* Position */
         for (j=0; j<numSegments[i]; j++)
         {
           HumanOutputSetup[i].HumKinematics[j][0]   = NOT_EDITABLE;  /* "X_CG"*/
           HumanOutputSetup[i].HumKinematics[j][1]   = NOT_EDITABLE;  /* "Y_CG"*/
           HumanOutputSetup[i].HumKinematics[j][2]   = NOT_EDITABLE;  /* "Z_CG"*/
           HumanOutputSetup[i].HumKinematics[j][3]   = NOT_EDITABLE;  /* "Roll"*/
           HumanOutputSetup[i].HumKinematics[j][4]   = NOT_EDITABLE;  /* "Pitch"*/
           HumanOutputSetup[i].HumKinematics[j][5]   = NOT_EDITABLE;  /* "Yaw"*/

         }
         for (j=0; j< MAXHVEJOINTS; j++)
         {
            HumanOutputSetup[i].Joints[j][0]   =    NOT_EDITABLE;
            HumanOutputSetup[i].Joints[j][1]   =    NOT_EDITABLE;
            HumanOutputSetup[i].Joints[j][2]   =    NOT_EDITABLE;
         }
      }
      return DataError;      /* No Humans in GeneralAnalysis */

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

Appendix F
EXAMPLE OF ExecuteCalcMethod()

```
/*-------------------------------------------------------------------------
 Structure for time positions of vehicle - used to interface with
   fortran common of same name  */
struct eventData
{
      int n;
      float tim[200];
      float pos[200][6];
}data_;
/*-------------------------------------------------------------------------*/

extern struct VehicleData *Vehicle;


INT ExecuteCalcMethod()
{

      /* ... Function that initializes and executes GeneralAnalysis.

         Called by:           HveMain
         Function Calls:      Initialize()
                              ComputeData()
      */

      Initialize();
      ComputeData();

} /* End of ExecuteCalcMethod() */


/* The following is where the initialization of variables would be placed if
   it was required...in this example this function does not have any code in it.

   It is simply listed to show the process */

void Initialize(void)
{
      /* Function that initializes program variables.

         Called by:           ExecuteCalcMethod()
         Function Calls:      (none)
      */


         ;

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

Appendix G
EXAMPLE OF ComputeData()

```
void ComputeData(void)
{
      /* This is the main calculation function of FortranProg.
         Called by:            ExecuteCalcMethod()
         Function Calls:       ftncalc_() - a FORTRAN function
                               WriteVehicleOutput()
                               SendHveOutput()    */
      /*------------------------ GLOBAL VARIABLES ---------------------
        INT istop;             Global error flag
      -------------------------- LOCAL VARIABLES ----------------------*/
        INT i,j,k;          /* Local indices                            */
      /*-----------------------------------------------------------------*/
      INT hstop;          /* program error key for sending HVE output tracks */
      INT numInBuff;      /* interpolation buffer - number in array to use */
      INT stat;           /* local status variable for monitoring file I/O */
      INT nveh;
      struct MyVehicleStructure veh[MAX_NUM_VEHICLES];
      struct MyHumanStructure hum[MAXHUMANS];
/*    ftncalc_() is a FORTRAN function which computes an array of vehicle
     postions as a function of time.  All based on the initial conditions set */

      ftncalc_();  /* call fortran program to perform physics */

      /* loop and load resulting data into output tracks */
      nveh = 0;
      for (i=0; i<data_.n; i++)
      {
        /* vehicle CG position and orientation */
         veh[nveh].s[0] = data_.pos[i][0]; /* x */
         veh[nveh].s[1] = data_.pos[i][1]; /* y */
         veh[nveh].s[2] = data_.pos[i][2]; /* z */
         veh[nveh].s[3] = data_.pos[i][3]; /* roll */
         veh[nveh].s[4] = data_.pos[i][4]; /* pitch */
         veh[nveh].s[5] = data_.pos[i][5]; /* yaw */

         /* Wheels and tires for vehciel */
         for (j=0; j<2; j++)
         {
            for (k=RIGHT; k<=LEFT; k++)
            {
               /* --- wheel hub data --- positions relative to the vehicle cg
                                         are the same as those initially defined.*/
               veh[nveh].wheel[j][k][0] = Vehicle[nveh].Wheel[j][k].Location.Coord[0];
               veh[nveh].wheel[j][k][1] = Vehicle[nveh].Wheel[j][k].Location.Coord[1];
               veh[nveh].wheel[j][k][2] = Vehicle[nveh].Wheel[j][k].Location.Coord[2];
               veh[nveh].wheel[j][k][3] = NOT_USED;
               veh[nveh].wheel[j][k][4] = NOT_USED;
               veh[nveh].wheel[j][k][5] = NOT_USED;
               veh[nveh].skid[j][k] = NOT_USED;/* ------- skid flag ------- */
               veh[nveh].tire[j][k][0] = NOT_USED;
               veh[nveh].tire[j][k][1] = NOT_USED;
               veh[nveh].tire[j][k][2] = NOT_USED;
            }
         }
      /*========== Load HVE output tracks and send ======================*/
         WriteVehicleOutput(data_.tim[i],nveh,veh[nveh]);

           if (hstop = SendHveOutput(numHumans,numVehicles)) istop = hstop;
      }

      return;
} /* End of ComputeData() */
```

Appendix H
EXAMPLE OF ShutDown()

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

INT ShutDown(void)
{
      /* Getting ready to leave Main Processing Module. Fill out all the
         reports and assign all the variables potentially required for
         warning messages.

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

        /* FILL OUT ALL REPORTS */


        /* CALL HveShutdown() SO IT CAN SEND REPORTS*/
        HveShutdown(istop);

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