ENGINEERING DYNAMICS COMPANY

RDF (ReadDataFile)

HVE Human Vehicle Environment

Markdown Edition — revised to describe the current version of the software

Contents

1. ReadDataFile — User's Manual
2. Chapter 1 — Program Description
3. Chapter 2 — Program Input
4. Chapter 3 — Calculation Method
5. Chapter 4 — Data File Format

ReadDataFile — User's Manual

For the HVE (Human Vehicle Environment) simulation environment

Updated Markdown edition, converted from the legacy ReadDataFile manual (manual printed August 17, 2004) and revised to match the behavior of the current HVE release.

(updated: the 2004 manual documents data-file format version 3.500. The current ReadDataFile calculation method reports program version 19.03, but still reads the version 3.200 and 3.500 data-file formats described in this manual. Within HVE, this manual is installed as `ReadDataFile.pdf` and opened from the Help > User Manuals menu.)

HVE is a trademark of Engineering Dynamics Company, LLC.

What is ReadDataFile?

ReadDataFile is an HVE-compatible physics program (calculation method) that imports time-based position and orientation data for vehicles and humans from a space-delimited ASCII data file, and turns that data into a normal HVE event. It is not a simulation: it interpolates externally generated motion data so that the standard HVE tools (Playback Editor, variable output tables, trajectory visualization) can be used with results from programs that have not been ported to HVE, from spreadsheets, or from any other calculation method.

Contents

- 1 — Program Description
- 2 — Program Input
- 3 — Calculation Method
- 4 — Data File Format (with examples)

Chapter 1 — Program Description

Overview

There are a number of computer simulation programs used in accident reconstruction analysis that have not yet been ported to the HVE computing environment. In an effort to assist the users of these programs, ReadDataFile has been produced. This HVE-compatible program accepts an ASCII data file that contains motion data for selected objects and loads the motion data into HVE. Once this has been accomplished, the standard HVE tools are available to generate a three-dimensional visualization of the data. The ReadDataFile event can also be combined with other HVE events in the Playback Editor.

This program is **not** a simulation, although it may be used to create visualizations of simulation results. The data file must contain time-dependent position data for the objects selected. ReadDataFile interpolates this data and produces an HVE event with the data. In addition, the program calculates the velocity and acceleration data for the objects and places this data in the variable output tables for easy viewing, printing, or graphing.

ReadDataFile can be used with vehicle and human objects. The data contained in the file does not have to be generated with a simulation program. The data may be generated with a spreadsheet or other calculation method, as determined by the requirements of the user.

Chapter 2 — Program Input

General Use

The ReadDataFile program is used to import time-based positional data into HVE to produce an HVE event representing the motion data contained in a space-delimited ASCII data file. Any type of motion can be defined in the data file, including motion from simulations as well as motion calculated in other ways. For example, a data file can be created that contains the time-position data from a vehicle overturn, a load coming off of a vehicle, a pole falling down, a helicopter landing, etc. Virtually any object motion can be generated using the ReadDataFile program and an appropriate ASCII data file.

The basic file format is shown in Figure 1. Each data file can have time-dependent position data for up to 10 different objects

(updated: the limit is 10 objects — an event containing more than 10 vehicles is rejected with the message "ReadDataFile is limited to 10 objects")

. The details of the data file format are contained later in this manual, but in general every data file contains six lines at the top of the file describing the data in the file. After these first six lines, the data file contains a time entry followed by all of the position and orientation data for every object for that instant in time. This is followed by the next time entry along with the position and orientation data for that instant in time, and so on.

Figure 1: General format of data file (`**` marks a line whose format changed from previous versions)

```
Version 3.500                    **
2D or 3D
Number of vehicles
Parent object numbers            **
Number of axles on each vehicle (one number for each vehicle; 3 vehicles = 3 separate
numbers)
Number of humans
Time0
     X0  Y0  Z0  Yaw0  Pitch0  Roll0
     . .
Time1
     X1  Y1  Z1  Yaw1  Pitch1  Roll1
     . .
Time2
     X2  Y2  Z2  Yaw2  Pitch2  Roll2
     . .
```

The six header lines come first; each time entry is then followed by one data line per object, and by any tire or human-segment lines (shown as `. .`) described in Chapter 4.

The data file format is free-form, meaning spaces are ignored (except to separate data values). Note that there should not be any commas or tab characters in the data file. The data should be separated by one or more spaces. In general it is a good idea to "structure" the data file like the examples shown in this manual, with the time entry on a line by itself followed by a line of data values for each of the separate objects. This is not required but aids in visually checking the data.

> **NOTE:** The initial positions must be specified. They can be entered at 0,0,0 and the program will move them to match the data file. The initial velocity has no effect.

Implementation

Prior to creating the ReadDataFile event, it is necessary to create the data file containing time-dependent position and orientation data for the various objects whose motion you wish to describe. The file should be of the format described in the previous section of this manual. Once that file is complete and in the correct format, save the file with a `.dat` extension in your HVE directory. For example, a valid file name would be `CarMotion.dat` and a valid path would be `C:\Program Files\HVE\CarMotion.dat`.

(updated: the data file must be placed in the HVE program directory — the folder that contains the `supportFiles` folder. ReadDataFile looks for it there and nowhere else. The data file name is limited to 30 characters, so keep it short.)

To create a ReadDataFile event:

1. Create the object or objects whose motion you wish to describe using either the Human Editor or the Vehicle Editor.
2. Click on the Event Editor button to enter the Event Editor.
3. Click on the (+) sign to add an event. In the Event Name field enter `File:FILENAME`, where *FILENAME* is the title of the motion data file. Figure 2 shows a sample Event Information dialog window. *(updated: the event name must begin with `File:` followed immediately by the data file name, with no space between them.)*
4. Select the objects in the Event Humans & Vehicles dialog box in the order that their motion data appears in the data file.
5. Set the Calculation Method type to *ReadDataFile* (shown as READDATAFILE in the method list).
6. Click OK to create the event.

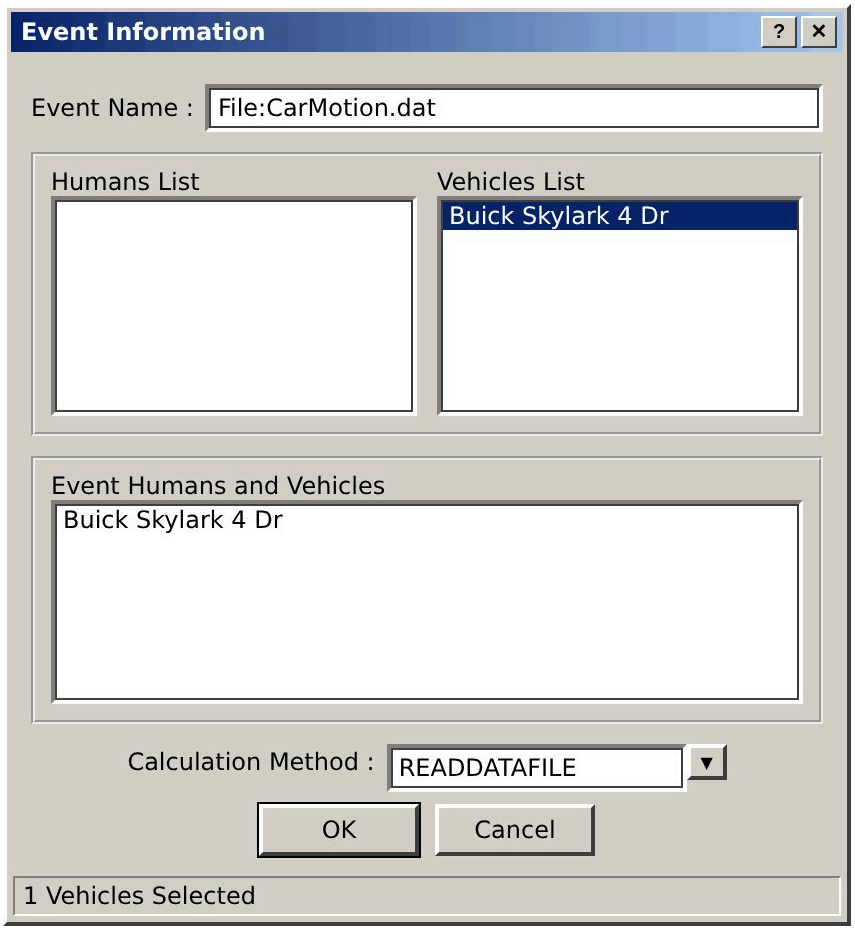


Figure 2: Event Information dialog window, showing an event named "File:CarMotion.dat" with a Buick Skylark 4 Dr selected and Calculation Method READDATAFILE.

If an error message is printed, such as the one shown in Figure 3 ("Bad or Incompatible Event Data"), there could be a problem with the format of the data in your motion data file, or the data file may not be in the HVE directory. Verify that the proper number of objects are selected and that the motion data file describes the motion of the same number of objects. Another option when troubleshooting an error message when executing a ReadDataFile event is to view the debug file in the HVE program directory using a text viewer. This file contains information regarding the operation of ReadDataFile when trying to access your data file and may indicate where the error lies.

(updated: the 2004 manual calls this debug file `ReadDataFileDebug.prn` in `C:\Program Files\HVE`. The current release writes it as `ReadDataDebugFile.prn` in the HVE program directory. The debug file echoes the expected basic file format, the header values as they are read, the number of data lines expected per time increment, and the ending time and time increment.)

Once the event is created without any error messages, set the initial position for all of the objects contained in the event to arbitrary values. Since the initial positions and orientations of the objects are contained in the data file, HVE will discard these arbitrary entries and use the data file values for the initial position and orientation in the ReadDataFile event.

Click on the execute button to execute the ReadDataFile event.



Figure 3: Error Message returned by ReadDataFile — an HVE alert dialog reading "Bad or Incompatible Event Data".

Chapter 3 — Calculation Method

Overview

ReadDataFile uses linear interpolation to calculate the position and orientation of objects between the designated time steps contained in the data file. The data file must contain time-dependent position and orientation data (i.e. X, Y, Z, Yaw, Pitch, Roll).

In addition, ReadDataFile differentiates the position data to produce velocity and acceleration data. **BE EXTREMELY CAREFUL WITH VELOCITY AND ACCELERATION DATA.** This program uses the interpolated position data, based on the Output Time Interval contained in the Simulation Controls dialog, to calculate velocity and acceleration. Therefore, if the data file contains finely spaced time data and the Output Time Interval is large compared to the data file time increment, then the program may "miss" the acceleration.

For example, consider a data file that contains position data at time = 0.0 seconds and then data every 0.1 second up to 2.0 seconds. Further assume that the data describes a vehicle traveling in a circle and that the position and orientation data at 2.0 seconds is the same as at 0.0 seconds. Therefore, at 1.0 seconds the vehicle is half way around the circle. If the Output Time Interval is chosen to be 1.0 seconds, then the vehicle will move in a straight line across the circle and return. If the Output Time Interval is 2.0 seconds the vehicle will not move at all. Thus, the user must make sure that the Output Time Interval is small enough to generate the correct motion. If possible, use an Output Time Interval that is equal to the smallest time increment that appears in the data file.

The same potential error may occur with velocity and acceleration data. If the Output Time Interval is too large compared to the time increments in the data file, then the velocity and accelerations will probably not be correct. This feature is implemented in an effort to assist in producing "rough" acceleration pulse data, but as with all computer programs... CHECK ALL OF YOUR DATA CAREFULLY.

(updated: the program interpolates between each successive pair of data-file entries and steps the simulation clock from 0 to the Max Time from the Simulation Controls dialog in steps of the Output Time Interval. Velocities are computed by finite differences of the interpolated positions from one output step to the next, and accelerations by finite differences of those velocities; at the first output step velocity and acceleration are reported as zero, and at the second step acceleration is still zero. Angular data is screened between successive file entries for discontinuities caused by quaternion-to-roll/pitch/yaw conversions in the originating program, and adjusted (±180° with pitch mirrored) when a discontinuity is detected. If the end of the data file is reached before Max Time, the event simply ends at the last file time.)

In addition, use care when selecting the precision of the input data. The input data should be entered in INCHES and DEGREES. Select the decimal accuracy of the data depending on the time increment of the data. In other words, if the data is in 0.001 second intervals, then 0.001 inches may be the amount of accuracy you need in your positional data.

GREAT CARE MUST BE USED WITH THE ORIENTATION DATA. In other words, in HVE +180 degrees yaw and -180 degrees yaw may look the same but will result in different motion. The positive yaw angle will produce a clockwise yaw movement while the negative yaw angle will produce a counter-clockwise yaw movement. THIS IS VERY IMPORTANT AND IF THIS BEHAVIOR IS NOT FULLY UNDERSTOOD THEN INVALID MOTION MAY BE PRODUCED.

Chapter 4 — Data File Format

Overview

ReadDataFile requires a certain format for any data file that is to be used to describe vehicle motion. It is made up of a number of header lines followed by the time-dependent position and orientation data for each object to be included in the ReadDataFile event. All data contained in the data file should be space delimited. The following portions of this chapter explain this format.

Header Format

The header of the ReadDataFile data file must contain six lines before any time-dependent data. These lines are used to specify the type of data that is contained in the file.

- **First Line** — contains the version number. The version number is used to determine possible options and data format changes from one version to another.

(updated: the line has the form `Version 3.500` — a word beginning with `V` or `v` followed by the version number. If this line is missing, the file is read as version 3.200 format, in which case the parent-object line (fourth line, below) must be omitted. The parent-object line is read only when the file version is 3.5 or later.)

- **Second Line** — contains a key word, either `2D` or `3D`, to specify whether the data will be 2-dimensional tire data or fully 3-dimensional tire data.

(updated: either `2D` or `2d` is accepted for two-dimensional data, but only the uppercase `3D` is recognized for three-dimensional data — always write the key in uppercase.)

The difference between 2D and 3D tire data is that 2D data only requires (and uses) the steer angle for all the axles specified along with the skid flag. Therefore, for 2D tire data only two data values are required for each specified axle for each time in the data file. For every axle there will be two lines of data, the first for the right side and the second for the left side. If only one axle is specified then it will reference the front axle. For a tractor truck with three axles, the first axle is the steer axle, the second axle is the front drive axle, and the third axle is the aft drive axle. If the number of axles is set at two for a three-axle vehicle then the steer axle and front drive axles would require two data lines each (right and left).

3D tire data contains more information. The displacement from normal position (x, y, z), the orientation from normal (steer, spin, camber or yaw, pitch, roll), the skid flag, and the contact location of the tire on the ground (cX, cY, cZ — used to draw skid) are all required for each time step. There are 10 data values required for each time step and each axle

side

for 3D tire data. Note that the ground contact (cX, cY, cZ) are relative to the world fixed coordinate system and the displacements and angles are with respect to the normal tire position.

(updated: a 2D tire line carries 2 values and a 3D tire line carries 10. The ten 3D values are read in the order: x y z yaw pitch roll skid-flag cX cY cZ. The x, y, z displacements are added to the wheel location defined in the Vehicle Editor; the contact-point z is raised 0.02 inches when drawn so the skid appears just above the road surface. The skid flag (0 = not skidding, 1 = skidding) and contact point are applied to both tires of a dual pair; for 2D data the contact points of dual tires are offset laterally from the wheel location by half the tire spacing.)

- **Third Line** — Number of vehicles that this data file has data for.

- **Fourth Line** — Indicates the parent object for each of the objects. This allows relative motion of one object with respect to another.

For instance, relative data can be used to specify a windshield wiper rotating on the "parent" object. In this case the relative motion is simply rotational and the position does not move relative to the parent, even though the wiper may be moving relative to the world coordinates. Use zero (0) to indicate motion relative to the world, or the number of the object in the object list for relative motion. For example, if there are three vehicles in the event, specifying a "1" will cause motion in the data file to be relative to the first vehicle. There should be an entry on this line for every vehicle in the event, and they may all be zero. Entries, as always, should be separated by spaces. Also, the parent object must be specified prior to the child object. In other words, the second vehicle CANNOT be specified relative to the third vehicle... the parent must be specified before the child.

(updated: the file is rejected with an error message if the first vehicle's parent is not 0 ("FIRST VEHICLE MUST REFERENCE THE FIXED COORD. SYSTEM") or if any vehicle's parent number is greater than its own position in the list ("Parent Vehicle MUST be defined before child vehicle"). For a child object, the interpolated position is transformed through the parent's position and orientation, and the child's yaw/pitch/roll are added to the parent's angles.)

- **Fifth Line** — Number of axles on each of the vehicles.

A number must be included for each vehicle in the ReadDataFile event. If the third line has the number "3" then there should be three numbers on this line. The first number indicates how many axles the first vehicle has, the second number on this line would be the number of axles for the second vehicle, etc. Again, these values should be separated by spaces. If there is no requirement to show the wheels moving at all then the number of axles can be set to zero. In most cases, there is no need to put any of the tire data in the data file because the visualization will show the tires in the normal position and orientation throughout the run (i.e. displacement and angles will all be set to zero).

- **Sixth Line** — Number of humans that this data file has data for.

Human data are specified differently than vehicles. For a human, the only position data required is for the pelvis segment. The orientation data (yaw, pitch, roll) is specified for the pelvis as well as the other fourteen mass segments. The first data line contains six data values (x, y, z, yaw, pitch, roll) and the next 14 lines contain only three data values, the orientations relative to the parent segments. Again, all data values should be separated by spaces.

(updated: humans are always read as 15 segments: one 6-value line for the lower torso (pelvis) followed by 14 3-value joint-orientation lines. Note also that the current release positions the human's main segment relative to the first vehicle's coordinate system, and assumes one human and one vehicle in the event.)

Data Values

Vehicle data specified in the data file is to be given with respect to the world coordinate system as defined in HVE. Times are indicated in seconds, positions in inches, and orientations in degrees. Vehicle-trailer combinations are treated like separate vehicles in ReadDataFile. This allows for detailed motion to be specified by the user. For example, a user can demonstrate a trailer becoming separated from its towing vehicle. This requires that the position and orientation of the trailer be specified just like any other vehicle (x, y, z, yaw, pitch, roll) all relative to the world-fixed coordinate system. This differs from using a vehicle-trailer combination in a physics module, such as SIMON, where the physics module determines that the towing vehicle and trailer are to be hitched together throughout the simulation.

If the value for Number of Axles is set to zero for any vehicle there will not be any data lines for those wheels reported in the Key Results window or the Playback Editor. When any wheel data is omitted in this way, the wheels will simply appear in the locations specified in the Vehicle Editor.

Human data is entered relative to the vehicle-fixed coordinate system.

(updated — order of values within a data line: each vehicle line is `X Y Z Yaw Pitch Roll`. Each time block consists of the time value followed by, for each vehicle in order: one vehicle line, then the tire lines (2 or 10 values each) for each axle, right side then left side; then, for each human: the 6-value pelvis line and 14 joint lines. The total number of data items per time block is limited to 100. Do not use commas or tabs: any entry that is not a number is treated as the end of the file, and the event stops at that point.)

Examples

For the purposes of the examples, the values contained within the bounding code blocks below are the actual contents of the ASCII data file. The commentary describes what the values within the blocks represent.

Example 1

This example will use the simplest data file that can be accepted into ReadDataFile. This data file contains the six required header lines and only two time entry lines for a single vehicle. In this simple case we will not produce tire data by setting the number of axles to zero. We will, however, set the tire code to 2D, use one vehicle, zero axles, and zero humans. The time dependent position data is outlined in Table Ex-1.

Table Ex-1: Example 1 positional and orientation data

Time (sec)	X (ft)	Y (ft)	Z (ft)	Yaw (deg)	Pitch (deg)	Roll (deg)
0.0	10.0	5.2	-2.1	90.1	0.0	0.0
5.0	115.0	7.2	-2.1	90.1	0.0	0.0

The corresponding data file is shown in Figure Ex-1. Recall that the positional data contained in the data file is in inches (the table above is in feet). Note that there are no commas in the data file and that the file is free-style format. The numbers do not have to be aligned in specific columns, there are no required decimals, etc.

Figure Ex-1: Completed data file for Example 1

```
Version 3.500
2D
1
0
0
0
0.0
   120.0  62.4  -25.2  90.1  0.0  0.0
5.0
   1380.0  86.4  -25.2  90.1  0.0  0.0
```

Line by line: line 1 = version number; line 2 = 2D tire data; line 3 = number of vehicles = 1; line 4 = parent objects; line 5 = number of axles = 0; line 6 = number of humans = 0; then the first time entry (t = 0.0 sec) followed by the position/orientation for vehicle 1, and the second time entry (t = 5.0 sec) followed by the position/orientation for vehicle 1.

(Figure reconstructed from the source scan.)

Example 2

This example is a little more complicated. It involves three vehicles but no axles. The time dependent data is shown in Table Ex-2.

Table Ex-2: Example 2 data set *(values in feet and degrees)*

Time (sec)	Vehicle 1 X, Y, Z, Yaw, Pitch, Roll	Vehicle 2 X, Y, Z, Yaw, Pitch, Roll	Vehicle 3 X, Y, Z, Yaw, Pitch, Roll
0.0	10.0, 5.2, -2.1, 90.1, 0.0, 0.0	20.0, 17.4, -2.5, 0.0, 0.0, 0.0	40.0, -15.0, -2.0, 180.0, 0.0, 0.0
5.0	115.0, 7.2, -2.1, 90.1, 0.0, 0.0	125.0, 17.4, -2.5, 0.0, 0.0, 0.0	-50.0, 25.0, 2.0, 180.0, 0.0, 0.0

The data file for this more complicated example is shown in Figure Ex-2.

Figure Ex-2: Data file for Example 2

```
Version 3.500
3D
3
0 0 0
0 0 0
0
0.0
   120.0   62.4  -25.2   90.1  0.0  0.0
   240.0  208.8  -30.0    0.0  0.0  0.0
   480.0 -180.0  -24.0  180.0  0.0  0.0
5.0
   1380.0   86.4  -25.2   90.1  0.0  0.0
   1500.0  208.8  -30.0    0.0  0.0  0.0
   -600.0  300.0   24.0  180.0  0.0  0.0
```

Line by line: line 1 = version number; line 2 = tire data key; line 3 = number of vehicles = 3; line 4 = parent objects (all world-fixed); line 5 = number of axles = 0 0 0; line 6 = number of humans = 0; then each time entry followed by one position/orientation line per vehicle.

(Figure and table reconstructed from the source scan; a few digits were illegible in the source scan and are approximate, but the structure is exact.)

Example 3

Example 3 is a complex example with two vehicles with two axles each. Using only the 2D tire data, only the steer angle and skid flag are used. For simplicity in this example the skid flags will be set to zero. The position and steer data are shown in Table Ex-3.

(Table Ex-3 — two vehicles, each with X, Y, Z, Yaw, Pitch, Roll plus front-axle right/left steer values at times 0.0 and 5.0 sec. Individual values are reproduced in the data file below.)

The data file for Example 3 is shown in Figure Ex-3 below.

Figure Ex-3: Data file for Example 3

```
2D
2
2 2
0
0.0
   120.0  62.4  -25.2  90.1  0.0  0.0
   8.0  0
   8.0  0
   0.0  0
   0.0  0
   240.0  208.8  -30.0  0.0  0.0  0.0
   0.0  0
   0.0  0
   0.0  0
   0.0  0
5.0
   1380.0  86.4  -25.2  90.1  0.0  0.0
   7.0  0
   8.0  0
   1.0  0
   0.0  0
   1200.0  200.0  -30.0  0.0  0.0  0.0
   7.0  0
   8.0  0
   1.0  0
   2.0  0
```

Line by line: line 1 = 2D tire data key; line 2 = number of vehicles = 2; line 3 = number of axles for each of the 2 vehicles; line 4 = number of humans = 0; then, for each time entry: the position/orientation line for vehicle 1 followed by its steer-and-skid-flag lines (axle 1 right side, axle 1 left side, axle 2 right side, axle 2 left side), then the same set of lines for vehicle 2.

(Figure reconstructed from the source scan; some steer-angle digits are approximate.)

(updated: note that the Example 3 file in the original manual has no `Version` line. Such a file is read as version 3.200 format, which is why it also has no parent-object line — the header is only five lines in that case.)

Example 4

Example 4 is a complex example with two vehicles (a truck with a blinker), two axles on the first vehicle, zero axles on the blinker object. Using only the 2D tire data, only the steer angle and skid flag are used. For simplicity in this example the skid flags will be set to zero. All motion for the first vehicle is relative to the world-fixed coordinate system while the motion for the blinker object is relative to the first vehicle.

The position and steer data are shown in Table Ex-4.

Table Ex-4: Example 4 data set *(positions in feet, angles in degrees; the blinker's values are relative to vehicle 1)*

Time (sec)	Veh 1 X	Veh 1 Y	Veh 1 Z	Veh 1 Yaw	Veh 1 Pitch	Veh 1 Roll	Axle 1 steer R/L	Axle 2 steer R/L	Blinker X	Blinker Y	Blinker Z	Blinker Yaw, Pitch, Roll	
0.00	10.0	5.2	-2.1	90.0	0.0	0.0	5.0 / 5.0	0.0 / 0.0	-10.5	2.5	0.5	0.0, 0.0, 0.0	Blinker hidden
1.00	10.0	15.2	-2.1	90.0	0.0	0.0	5.0 / 5.0	0.0 / 0.0	-10.5	2.5	0.5	0.0, 0.0, 0.0	Blinker hidden
1.01	10.0	15.3	-2.1	90.0	0.0	0.0	5.0 / 5.0	0.0 / 0.0	-10.7	2.5	0.5	0.0, 0.0, 0.0	Blinker shown
2.00	10.0	25.2	-2.1	90.0	0.0	0.0	0.0 / 0.0	0.0 / 0.0	-10.7	2.5	0.5	0.0, 0.0, 0.0	Blinker shown
2.01	10.0	25.3	-2.1	90.0	0.0	0.0	0.0 / 0.0	0.0 / 0.0	-10.5	2.5	0.5	0.0, 0.0, 0.0	Blinker hidden

In this case the second vehicle should be of "type" Fixed Barrier so that no tires are shown. The geometry file should be a small disc or other shape of the correct size for a blinker. The blinker color should be set correctly and the emissiveColor parameter should be set in the geometry file. Essentially, the motion file "hides" the blinker inside the vehicle geometry, then moves it rearward (relative to the parent vehicle) very quickly.

The data file for Example 4 is shown in Figure Ex-4. Because relative motion is used, the file must be version 3.5 or later so that the parent-object line is present.

Figure Ex-4: Data file for Example 4

```
Version 3.500
2D
2
0 1
2 0
0
0.0
   120.0   62.4  -25.2  90.0  0.0  0.0
   5.0  0
   5.0  0
   0.0  0
   0.0  0
  -126.0   30.0    6.0   0.0  0.0  0.0
1.0
   120.0  182.4  -25.2  90.0  0.0  0.0
   5.0  0
   5.0  0
   0.0  0
   0.0  0
  -126.0   30.0    6.0   0.0  0.0  0.0
1.01
   120.0  183.6  -25.2  90.0  0.0  0.0
   5.0  0
   5.0  0
   0.0  0
   0.0  0
  -128.4   30.0    6.0   0.0  0.0  0.0
2.0
   120.0  302.4  -25.2  90.0  0.0  0.0
   0.0  0
   0.0  0
   0.0  0
   0.0  0
  -128.4   30.0    6.0   0.0  0.0  0.0
2.01
   120.0  303.6  -25.2  90.0  0.0  0.0
   0.0  0
   0.0  0
   0.0  0
   0.0  0
  -126.0   30.0    6.0   0.0  0.0  0.0
```

Line by line: line 1 = version number; line 2 = 2D tire data; line 3 = number of vehicles = 2; line 4 = parent objects (vehicle 1 relative to the world, vehicle 2 — the blinker — relative to vehicle 1); line 5 = number of axles = 2 and 0; line 6 = number of humans = 0. Each of the five time entries is then followed by the position/orientation line for vehicle 1, its four steer-and-skid-flag lines (axle 1 right side, axle 1 left side, axle 2 right side, axle 2 left side), and the position/orientation line for the blinker relative to vehicle 1. Because the blinker has zero axles, no tire lines follow it.

The flash comes from the two closely spaced time pairs: between 1.0 and 1.01 seconds the blinker moves 2.4 inches rearward, out of the truck body and into view, and between 2.0 and 2.01 seconds it moves back inside. Shorten or repeat

these pairs to change the flash rate.

(Table Ex-4 and Figure Ex-4 have been reconstructed from a degraded scan in the original manual. The structure, units and the motion they describe are as in the original; a few individual digits were illegible and have been chosen to be consistent with the legible values.)