



## **ADAMS/Car Training Guide**

**VERSION 11.0**

PART NUMBER  
110CARTRG-02

Visit us at: [www.adams.com](http://www.adams.com)

U.S. Government Restricted Rights: If the Software and Documentation are provided in connection with a government contract, then they are provided with RESTRICTED RIGHTS. Use, duplication or disclosure is subject to restrictions stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Mechanical Dynamics, Incorporated, 2300 Traverwood Drive, Ann Arbor, Michigan 48105.

The information in this document is furnished for informational use only, may be revised from time to time, and should not be construed as a commitment by Mechanical Dynamics, Incorporated. Mechanical Dynamics, Incorporated, assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

This document contains proprietary and copyrighted information. Mechanical Dynamics, Incorporated permits licensees of ADAMS® software products to print out or copy this document or portions thereof solely for internal use in connection with the licensed software. No part of this document may be copied for any other purpose or distributed or translated into any other language without the prior written permission of Mechanical Dynamics, Incorporated.

©2001 by Mechanical Dynamics, Incorporated. All rights reserved. Printed in the United States of America.

ADAMS® is a registered United States trademark of Mechanical Dynamics, Incorporated.  
All other product names are trademarks of their respective companies.



# CONTENTS

## **Welcome to ADAMS/Car Training 7**

- A Brief History of ADAMS 8
- About Mechanical Dynamics 9
- Referencing Online Guides 10
- Getting Help at Your Job Site 12
- A Review of Basic ADAMS Terminology 13

## **Introducing ADAMS/Car 15**

- Motivation for Using ADAMS/Car 16
- User Modes 20
- Database Structure—A Directory Hierarchy 21
- Configuration File 22
- Workshop 1—Open and Run an Assembly 23

## **Basic Concepts 29**

- Data Hierarchy 30
- Test Rig 32
- Major and Minor Roles 33
- Naming Convention 34
- Workshop 2—Templates versus Subsystems 35

## **Creating and Adjusting Subsystems 41**

- Creating Subsystems 42
- Adjusting Hardpoints 43
- Adjusting Parameter Variables 44
- Adjusting Mass Properties 45
- Adjusting Springs and Dampers 46
- Workshop 3—Creating and Adjusting Suspensions 48

## **Using the Curve Manager 55**

- Property File Types 56
- Creating Property Files 57
- Modifying an Existing Property File 60
- Plot versus Table 61
- Workshop 4—Modifying Springs with the Curve Manager 62

## **Creating and Simulating Suspensions 65**

- Creating Suspension Assemblies 66
- Half-Vehicle Analyses 67
- Suspension Parameters 68
- Creating Loadcases 69
- Warning Messages 70
- Files Produced by Analyses 71
- Workshop 5—Running Suspension Analyses 72

## **Creating and Simulating Full Vehicles 73**

- Creating Full-Vehicle Assemblies 74
- Shifting Subsystems 75
- Updating Subsystems 76
- Updating Assemblies 77
- Full-Vehicle Analyses 78
- Adjusting Mass Automatically 80
- Workshop 6—Running Full-Vehicle Analyses 81

## **Driving Machine 83**

- Standard Driver Interface (SDI) and Driving Machine 84
- Why Use SDI? 85
- Creating Inputs for SDI 86
- Creating .dcf and .dcd Files 88
- Workshop 7—Editing .dcf and .dcd Files 98

## **Plot Configuration Files 105**

- What is a Plot Configuration File? 106
- Workshop 8—Creating Plot Configuration Files 107

## **Parameterization 109**

- Parameterization in ADAMS/Car 110
- Creating Hardpoints 111
- Creating Construction Frames 113
- Location Parameterization 115
- Orientation Parameterization 120

# CONTENTS...

---

## **Building Templates 127**

- Template Overview 128
- Template Topology 129
- File Architecture 130
- Building a New Template 132
- Types of Parts 133
- Rigid Bodies (Parts) 134
- Flexible Bodies (Parts) 135
- Geometry 136
- Attachments (Joints and Bushings) 137
- Springs 138
- Dampers 140
- Bumpstops and Reboundstops 141
- Suspension Parameter Array 142
- General Advice 143
- Workshop 9—Template-Builder Tutorial 144

## **Communicators 145**

- Types of Communicators 146
- Classes of Communicators 147
- Communicator Roles 149
- Naming Communicators 150
- Matching Communicators During Assembly 151
- Matching Communicators with Test Rigs 153
- Workshop 10—Getting Information About Communicators 155

## **Using Flexible Bodies 159**

- Flexible Body Overview 160
- Limitations of Flexible Bodies 161
- Getting Flexible Bodies 162
- Workshop 11—Flex Tutorial 163

## **Requests 167**

- Creating New Requests 168
- Types of Requests 169

## **Tires 171**

Tire Overview 172

ADAMS/Tire Modules 173

Tire Models 175

Tire Analyses 176

Workshop 12—Building a Wheel Template 177

## **Exploring Templates 181**

Investigating Templates 182

Understanding Templates 183

About the Database Navigator 184

Workshop 13—Exploring and Completing Templates 186

## **Additional Applications 193**

Conceptual Suspension Module and Driveline 194

Linear and Controls 196

Insight and Hydraulics 197

Vibration and Durability 198

Workshop 14—Using ADAMS/Linear with ADAMS/Car 199

## **Workshop 15—Full-Vehicle Assembly 205**

## **Example Analyses 213**

Types of Analyses 214

Gather Data for the Model 215

Packaging Analysis on Suspension 216

Kinematic Analysis on Suspension 217

Suspension-Compliance Analysis 219

Static-Loading Durability Analysis 220

Dynamic-Loading Durability Analysis 222

Front Suspension Analyses 223

Full-Vehicle Design and Analysis 224

## **Four-Post Vertical Excitation Test 225**

## **ADAMS/Car Files 259**



# WELCOME TO ADAMS/CAR TRAINING

Welcome to ADAMS/Car training. In this course, you learn how to use ADAMS/Car to create, catalog, and analyze suspension and vehicle assemblies.

## What's in this section:

- [A Brief History of ADAMS, 8](#)
- [About Mechanical Dynamics, 9](#)
- [Referencing Online Guides, 10](#)
- [Getting Help at Your Job Site, 12](#)
- [A Review of Basic ADAMS Terminology, 13](#)

# A Brief History of ADAMS

---

ADAMS: Automatic Dynamic Analysis of Mechanical Systems.

Technology was implemented about 25 years ago.

Mechanical Dynamics Incorporated formed by researchers who developed the base ADAMS code at University of Michigan, Ann Arbor, MI, USA.

Large displacement code.

Systems-based analysis.

Original product was ADAMS/Solver, an application that solves nonlinear numerical equations. You build models in text format and then submit them to ADAMS/Solver.

In the early 90's, ADAMS/View was released, which allowed users to build, simulate, and examine results in a single environment.

Today, industry-specific products are being produced, such as ADAMS/Car, ADAMS/Rail, and ADAMS/Engine.



# About Mechanical Dynamics

---

Find a list of ADAMS products at:

<http://www.adams.com/mdi/product/modules.htm>

Learn about the ADAMS – CAD/CAM/CAE integration at:

<http://www.adams.com/mdi/product/partner.htm>

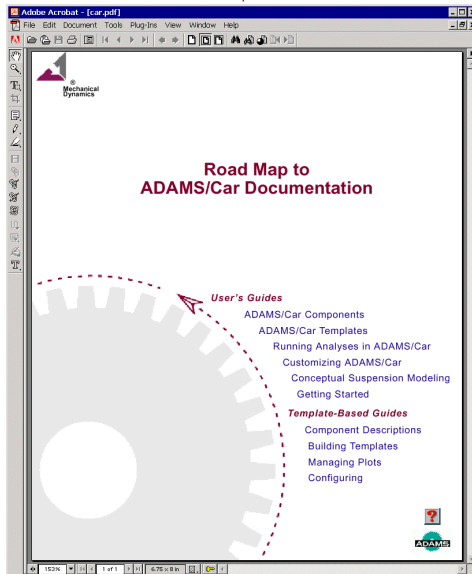
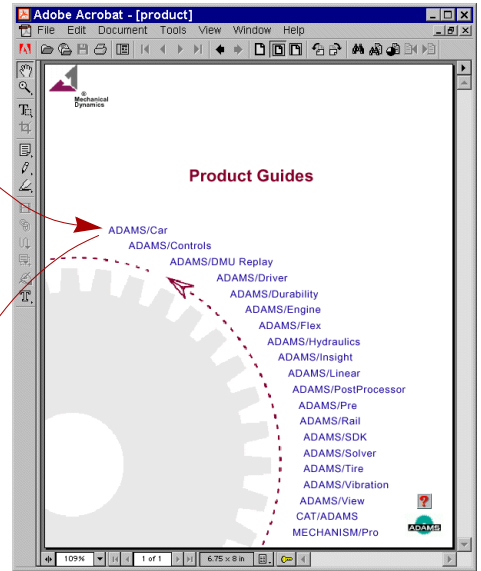
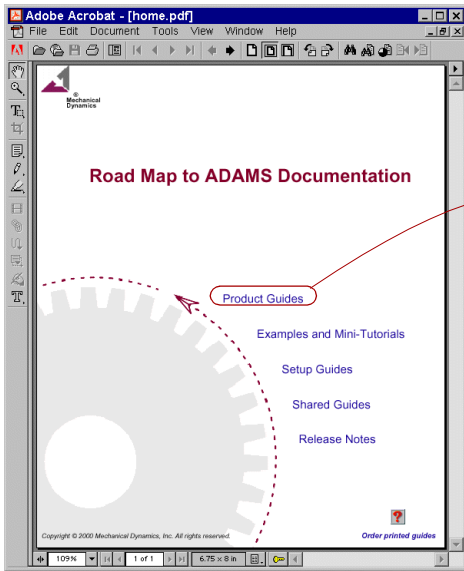
Find additional training at:

<http://support.adams.com/training/training.html>

...or your local support center.

# Referencing Online Guides

## Referencing the ADAMS/Car online guides



# Searching Online Guides

## Doing global searches on any online ADAMS guide

The process involves the following steps:

- Open the Adobe Acrobat window showing the **Product Guides** menu.
- Click the search icon in the toolbar to open the **Adobe Acrobat Search** dialog box.
- Enter the search term **toe angle** and click **Search**.
- The **Search Results** dialog box appears, showing 7 documents found. Select **Running Analyses in ADAMS/Car**.
- Click **View** to open the document in Adobe Acrobat.

The final document, **Running Analyses in ADAMS/Car**, contains the following information:

**Running Analyses in ADAMS/Car**  
Understanding Suspension Characteristic Calculations

**Toe Angle**

**Description**  
The **toe angle** is the angle between the longitudinal axis of the vehicle and the line of intersection of the wheel plane and the road surface. ADAMS/Car reports **toe angle** in radians. It is positive if the wheel front is rotated in towards the vehicle body.

**Request Names**

- toe\_angle\_left
- toe\_angle\_right


**Inputs**  
Wheel center axis unit vectors - left and right

**Method**  
ADAMS/Car uses the direction cosines in the x- and y-directions of the wheel center axis relative to the road to calculate **toe angle**, such that:  
$$\text{Left toe} = \tan^{-1} (\text{DCOSX}/\text{DCOSY})$$
$$\text{Right toe} = \tan^{-1} (-\text{DCOSX}/\text{DCOSY})$$

# Getting Help at Your Job Site

## Online guides

### Access help on help from:

- Help menu of any ADAMS product
- Help tool  on the Documentation Road Maps

## Knowledge base

Go to <http://support.adams.com/kb>

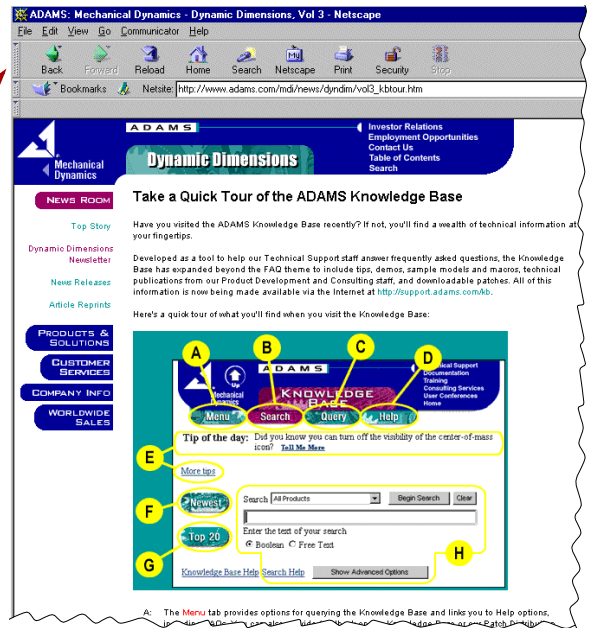
For a quick tour, go to: [http://www.adams.com/mdi/news/dyndim/vol3\\_kbtour.htm](http://www.adams.com/mdi/news/dyndim/vol3_kbtour.htm)

## ASK email-based users group

Go to [http://support.adams.com/support/tech\\_ask.html](http://support.adams.com/support/tech_ask.html)

## Consulting services

Go to: <http://support.adams.com/support/cnslsrv.html>



## Technical support

To find your support center, go to: <http://support.adams.com/support/suppcent.html>

To read the Service Level Agreement, go to: [http://support.adams.com/support/sla\\_agree.html](http://support.adams.com/support/sla_agree.html)

# A Review of Basic ADAMS Terminology

---

## ADAMS/Solver

The solution engine.

## ADAMS/Solver dataset (\*.adm)

The ASCII file submitted to ADAMS/Solver.

## ADAMS/Solver command (\*.acf)

An ASCII file that contains commands to control how ADAMS/Solver runs the model.

## ADAMS/Solver output files

- **Graphics** (\*.gra) - Information about how graphics work.
- **Request** (\*.req) - Contains output for a specific set of results.
- **Results** (\*.res) - Contains results for every entity. This file is too big, and is not produced by ADAMS/Car as default; you can, however, change ADAMS/Car to print this.
- **Message** (\*.msg) - Information about the solver/simulation/problems.
- **Output** (\*.out) - Output including initial conditions and request, and content can depend on output specifications.



# 1

## INTRODUCING ADAMS/CAR

This module discusses the advantages of using ADAMS/Car, as well as the organization of the basic files.

### What's in this module:

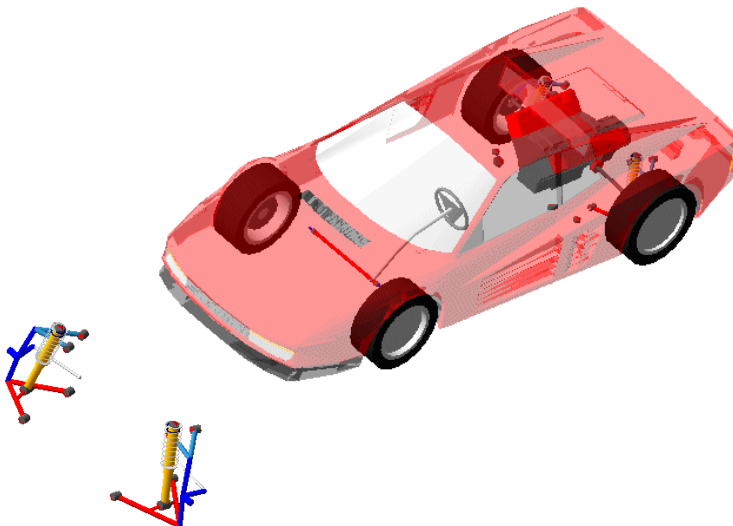
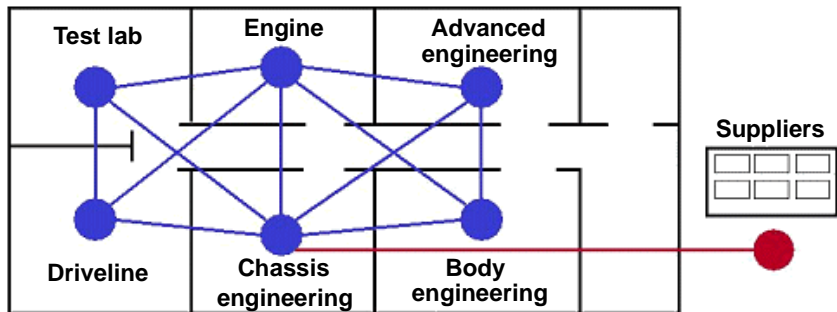
- Motivation for Using ADAMS/Car, 16
- User Modes, 20
- Database Structure—A Directory Hierarchy, 21
- Configuration File, 22
- Workshop 1—Open and Run an Assembly, 23

# Motivation for Using ADAMS/Car

Bridges departments by sharing models and data

Facilitates quick subsystem changes

Templates





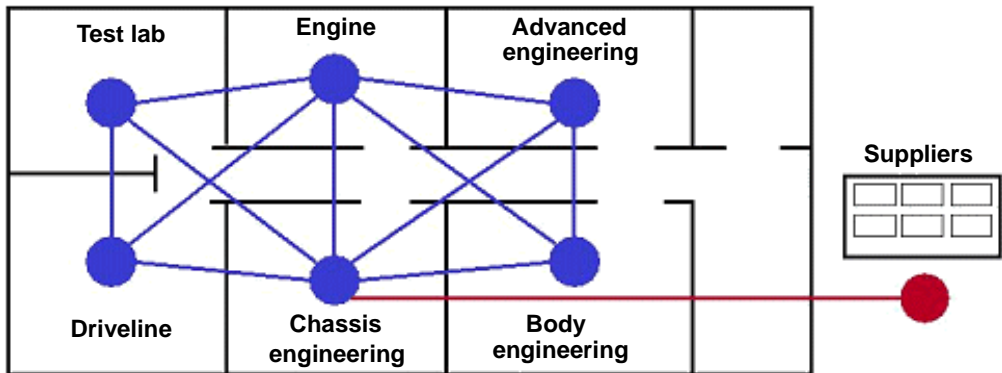
# Motivation for Using ADAMS/Car...

## Bridges departments by sharing models and data

Different departments can work with the same database, which minimizes data loss.

Facilitates quick subsystem changes

Templates



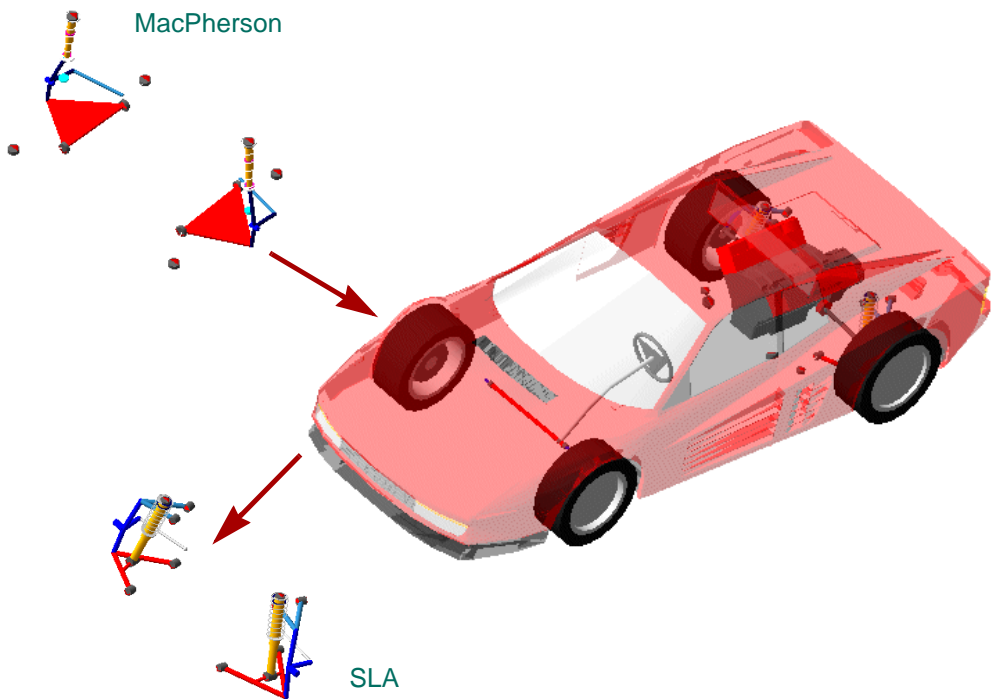
# Motivation for Using ADAMS/Car...

Bridges departments by sharing models and data

## Facilitates quick subsystem changes

You can easily replace one subsystem without changing any other part of the vehicle.

## Templates



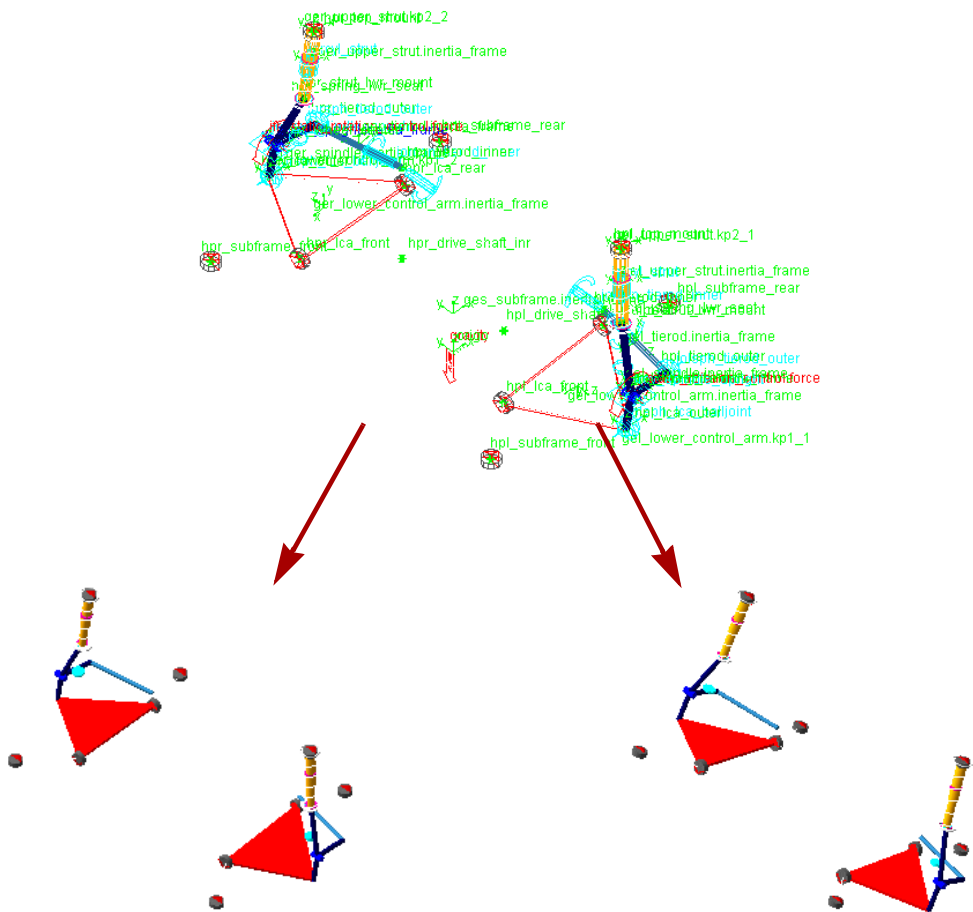
## Motivation for Using ADAMS/Car...

## Bridges departments by sharing models and data

## Facilitates quick subsystem changes

## Templates

Allow you to tailor one system for multiple vehicles.



# User Modes

---

Within the ADAMS/Car configuration file (.acar.cfg), the particular application of ADAMS/Car is specified as either standard user mode and expert user mode.

## Expert user (Template Builder and Standard Interface)

- Allows creation of building-block of ADAMS/Car, templates, with access to Template Builder
- For experienced ADAMS users
- Access to all ADAMS modeling entities

## Standard user (Standard Interface only)

- Specifically for designers and testing engineers
- Use libraries from the ADAMS/Car database to easily create vehicle (sub)assemblies
- Simulation environment tailored to automotive standards

# Database Structure—A Directory Hierarchy

A database is a collection of directories stored on the hard drive. The top directory, which has the extension `.cdb`, stores a number of tables (directories). Each table is a placeholder for model information.

## Three types of databases:

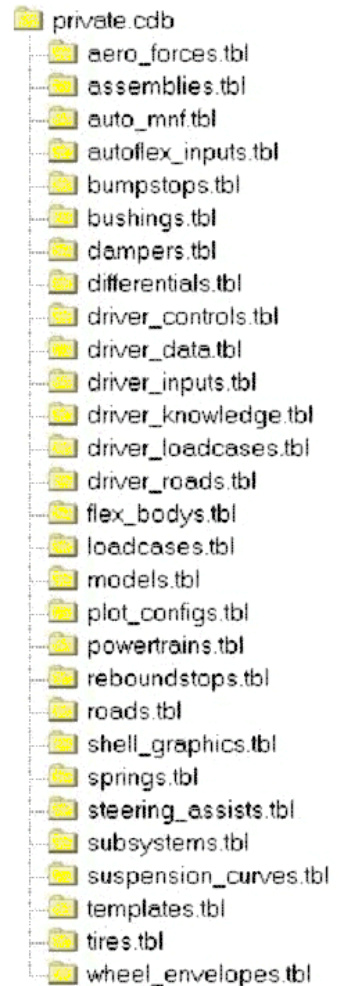
- **Shared** - Common to all users, provided by MDI with example files.
- **Private** - User workspace (created by ADAMS/Car in your \$HOME directory).
- **User** - User/site specific.

The databases are defined in private `.acar.cfg` or common `acar.cfg`.

## No limitations on number of databases

## Each project should have a separate database

## You can only save to one database at a time



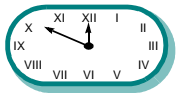
# Configuration File

For each user, ADAMS/Car creates a private configuration file, named `.acar.cfg` (Notice the first period, which distinguishes it from the common `acar.cfg`). This file is located in the users HOME directory and defines personal settings as:

- User mode (expert versus standard)
- Personal databases and tables
- Default property files
- Default writable database
- Database search order
- Orientation of global reference frame
- Other preferences

```
!-----!  
!***** ADAMS/Car Configuration File *****!  
!  
!-----!  
! - List of personal environment variables  
!-----!  
ENVIRONMENT MDI_ACAR_USERMODE      expert  
ENVIRONMENT MDI_CAR_RENDER          sshaded  
ENVIRONMENT MDI_ACAR_VEHICLE_REAR   1, 0, 0  
ENVIRONMENT MDI_ACAR_VEHICLE_LEFT   0, -1, 0  
!  
!-----!  
! - List of personal database directories  
!      Database name  Path of database  
!-----!  
DATABASE      private      D:\private.cdb  
DATABASE      dbase_1      D:\dbase_1.cdb  
DATABASE      dbase_2      D:\dbase_2.cdb  
DEFAULT_WRITE_DB  private  
!  
!-----!  
! - Desired database search order  
!-----!  
DATABASE_ORDERprivate, dbase_1, dbase_2, shared  
!
```

# Workshop 1—Open and Run an Assembly



This workshop takes about one half hour to complete.

## Problem statement


This workshop introduces you to a couple of typical ADAMS/Car simulations. ADAMS/Car basically runs either suspension or full-vehicle analyses. Here, you will perform one of each type: an ISO lane change for a full vehicle, and a parallel wheel travel for a front suspension. You will also add a trace marker to see the movement of particular parts in the model.

## Setting Up Your Session

### To create a working directory:

- Depending on the platform you're on, do one of the following:
  - ◆ On UNIX:
    - ◆ To start ADAMS/Car in your home directory, open an UNIX shell and type `cd`.
    - ◆ To create a directory named `acar`, type `mkdir acar`.
    - ◆ To move to the new directory, type `cd acar`.
  - ◆ On Windows:
    - ◆ On your hard drive, create a new folder `acar`. For example, `C:\acar`.

### To start ADAMS/Car:

- Depending on the platform you're on, do one of the following:
  - ◆ At your UNIX shell prompt, type `adams11`.
    - ◆ From the toolbar, select the **ADAMS/Car** tool .
  - ◆ On Windows, from the **Start** button, point to **Programs**, point to **ADAMS 11.0**, point to **ACAR**, and then select **ADAMS - Car (view)**.

The Welcome dialog box appears.

# Workshop 1—Open and Run an Assembly...

---

## To toggle to Standard Interface:

- From the **Welcome** dialog box, select **Standard Interface**, and then select **OK**. (Sometimes the Welcome dialog box contains the option to select a mode, and other times it does not. This depends on the configuration file.) To run analyses, you must be in the Standard Interface mode.
- Once in an ADAMS/Car session, you can toggle between modes by selecting ADAMS/Car Standard Interface from the Tools menu. If ADAMS/Car Template Builder is listed, then you are already in the Standard Interface mode.

## To create a new database and set it as the writable database:

- 1 From the **Tools** menu, point to **Database Management**, and then select **Create Database**.  
The Create New Database dialog box appears.
- 2 In the **Database Name** text box, enter **acar\_training**.
- 3 In the **Database Path** text box, enter the desired path. The database name is an alias for the Database Path, which needs to be explicitly defined. For example:
  - ◆ On NT: c:\dir1\dir2\acar\_training.cdb
  - ◆ On UNIX: /dir1/dir2/acar\_training.cdb
- 4 Select **OK**.
- 5 From the **Tools** menu, point to **Database Management**, and then select **Set Default Writable**, and make sure the **Database Name** is set to **acar\_training** (select the down arrow and then select **acar\_training**).
- 6 Select **OK**.



# Workshop 1—Open and Run an Assembly...

---

## Simulating a full-vehicle assembly

You first open a full-vehicle assembly and then perform a full-vehicle, ISO lane-change analysis with Driving Machine. You then investigate the results by animating the assembly. The animation is based on the results of your analysis.

### To open a full-vehicle assembly:

- 1 From the **File** menu, point to **Open**, and then select **Assembly**.
- 2 Right-click the **Open Assembly** text box, select `<shared>\assemblies.tbl`, and then select **MDI\_Demo\_Vehicle.asy**.
- 3 Select **OK**.

In the Message window, ADAMS/Car informs you when the assembly is ready.

- 4 Close the **Message** window.

### To perform an analysis:

- 1 From the **Simulate** menu, point to **Full-Vehicle Analysis**, point to **Course Events**, and then select **ISO Lane Change**.
- 2 In the **Output Prefix** text box, enter **workshop1a**.
- 3 In the **Initial Velocity** text box, enter **70**.
- 4 Select **OK**.


In the Message window, ADAMS/Car informs you about the progress of the analysis and when the simulation is complete.

- 5 Close the **Message** window.

# Workshop 1—Open and Run an Assembly...

---

## To investigate the results:

- 1 From the **Review** menu, select **Animation Controls**.
- 2 To animate the assembly, select the **Play** tool .
- 3 Zoom out to see more of the road grid:
  - Type a lowercase **z**.
  - Hold down the left mouse button, and do either of the following:
    - ◆ To enlarge the display of the assembly, or zoom in, move the cursor up.
    - ◆ To shrink the display of the assembly, or zoom out, move the cursor down.
  - To exit zoom mode, release the mouse button.

## To add a trace marker:

- 1 Change **No Trace** to **Trace Marker**.
- 2 Right-click the **Trace Marker** text box, point to **Marker**, and then select **Browse**.

The Database Navigator appears.
- 3 Double-click **MDI\_Demo\_Vehicle**, double-click **TR\_Body**, double-click **ges\_chassis**, and then select **cm**.

## To follow the car in the animation:

- 1 Change **Fixed Base** to **Base Part**.
- 2 Right-click the **Base Part** text box, point to **Body**, and then select **Pick**.
- 3 Move the cursor over the vehicle and select **ges\_steering\_wheel**.
- 4 Select **Play**.

The camera should move with the car, as the white line traces the path of the body marker.

# Workshop 1—Open and Run an Assembly...

---

## Simulating a suspension assembly

You simulate a suspension assembly in the same way you simulated the full-vehicle assembly.

### To open a suspension assembly:

- 1 From the **File** menu, point to **Open**, and then select **Assembly**.
- 2 Right-click the **Open Assembly** text box, select `<shared>\assemblies.tbl`, and then select `mdi_front_vehicle.asy`.
- 3 Select **OK**.

In the Message window, ADAMS/Car informs you when the vehicle assembly is ready.

- 4 Close the **Message** window

### To perform a parallel wheel travel suspension analysis:

- 1 From the **Simulate** menu, point to **Suspension Analyses**, and then select **Parallel Wheel Travel**.
- 2 Set up the analysis:
  - **Output Prefix:** workshop1b
  - **Number of Steps:** 10
  - **Bump Travel:** 100
  - **Rebound Travel:** -100
- 3 Select **OK**.
- 4 Close the **Message** window.

### To review the results by animating your assembly:

- 1 From the **Review** menu, select **Animation Controls**.
- 2 Select the **Play** tool.
- 3 Zoom out to see more of the road grid.

# Workshop 1—Open and Run an Assembly...

---

## To add a trace marker:

- 1 Change **No Trace** to **Trace Marker**.
- 2 Right-click the **Trace Marker** text box, point to **Marker**, and then select **Browse**.  
The Database Navigator appears.
- 3 Double-click **Front\_Suspension**, double-click **gel\_spindle**, and then select **cm**.
- 4 Select **Play**.
- 5 Zoom in to look for the white line that traces the path of the **gel\_spindle.cm** marker.

In this module, you learn how to create models in ADAMS/Car. This module describes the relationships and differences between templates, subsystems, and assemblies, which define ADAMS/Car models.

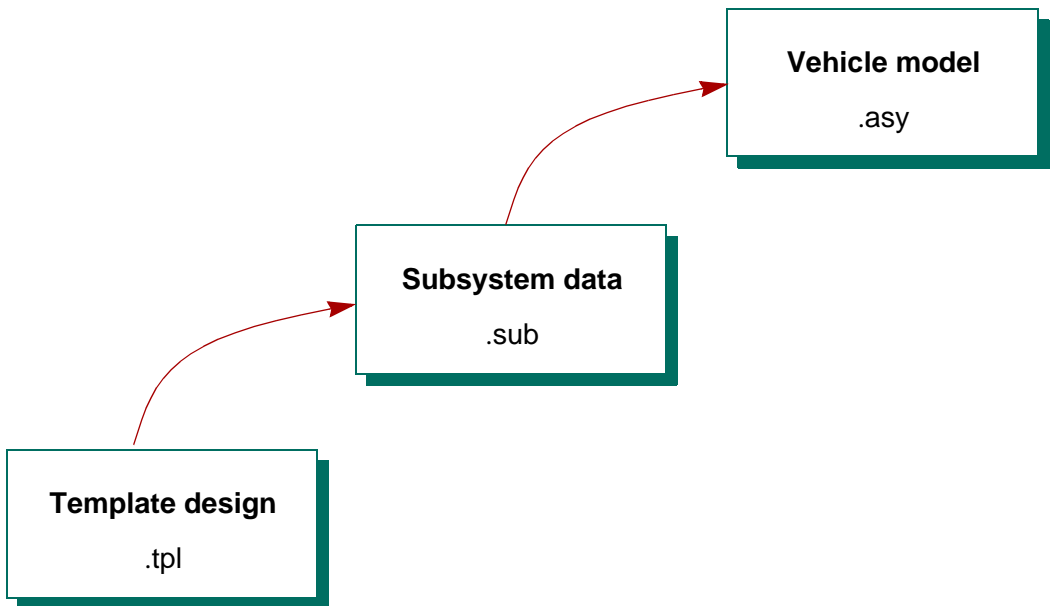
### What's in this module:

- Data Hierarchy, 30
- Test Rig, 32
- Major and Minor Roles, 33
- Naming Convention, 34
- Workshop 2—Templates versus Subsystems, 35

# Data Hierarchy

## Three levels of files build up a vehicle model (full or half vehicle):

- **Template** - Defines vehicle sub-assemblies topology (that is, how the parts and joints fit together in the model, how information is transmitted, and so on). For example, a template could be a suspension type, which can be defined either as front and/or rear.
- **Subsystem** - A mechanical model that references a template and tailors it by supplying parameters that adjust the template (for example, locations that define part dimensions and spring stiffness). These models are usually a major system of your vehicle, for example, front suspension, steering system, and body. The subsystem is a specific instance of the template in which the user has defined new hardpoint positions and property files.
- **Assembly** - A list of subsystems and a single test rig combined in a vehicle or suspension assembly. A test rig is necessary to provide an actuation, in your model, for analysis.

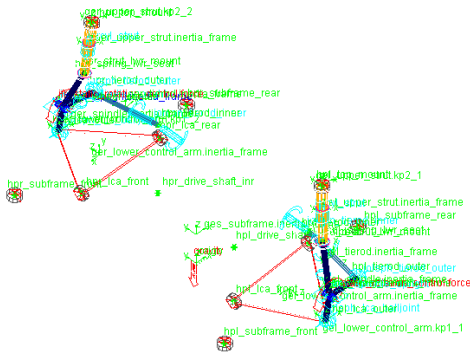


# Data Hierarchy...

The figure shows how a subsystem is created from a template. The template holds default geometry and topology. The subsystem is a specific instance of the template in which the user has defined new model parameters, such as hardpoint positions, property files, and mass properties.

## Subsystem File Creation

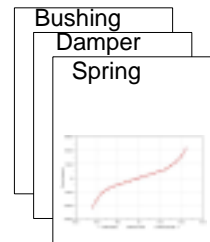
### Default MacPherson Strut Template



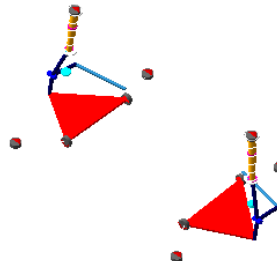
### Specific Mass Properties

### New Geometry Points

### Specific Property Files



### MacPherson Subsystem

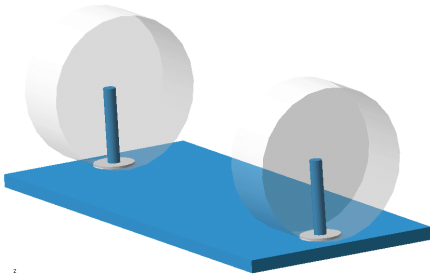


# Test Rig

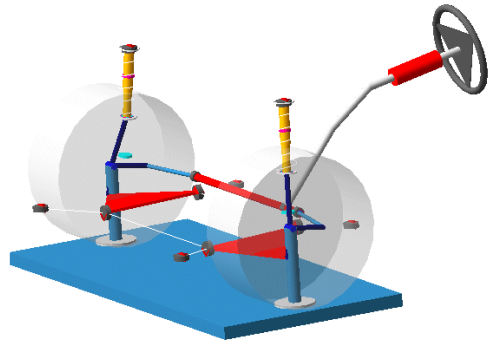
A test rig in ADAMS/Car is the part of the model that imposes motion on the vehicle. Depending on the model and event, different test rigs must be used.

A test rig is a special subsystem that is connected to all of the other subsystems that make up your model, forming an assembly. The figure on the left shows the suspension test rig alone. With the suspension subsystems, it would look like the figure on the right (an assembly).

Suspension test rig alone



Suspension test rig assembled with suspension and steering subsystems





# Major and Minor Roles

---

ADAMS/Car uses *major* and *minor* roles to create a valid assembly. Major and minor roles define the location of the subsystem within the assembly.

Every template (and therefore the subsystem that is created from that template) has a defined major role: suspension, steering, body, anti-roll bar, wheel, and so on.

When a subsystem is created, the standard user defines the subsystem minor role: *front*, *rear*, *trailer*, or *any*. This enables the same suspension template to be used for both a front and rear suspension.

To create a valid suspension assembly, the minimum requirement is a suspension subsystem and the ADAMS/Car suspension test rig.

To create a valid vehicle assembly, the minimum requirement is a front suspension, a rear suspension, front and rear wheels, a body, and a steering subsystem.

# Naming Convention

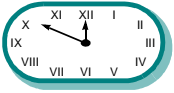
---

All ADAMS/Car entities are named after a naming convention. The first three letters of a given entity identify the type and the symmetry rule

## Examples:

- gel\_arm: General\_Part\_Left\_....
- hps\_lcs\_front: Hard\_Point\_Single\_...
- bkl\_mount: Bushing\_Kinematic\_Left\_...
- nsr\_main\_spring: Non-linear\_Spring\_Right\_...
- pvs\_toe\_angle: ParameterVariable\_Visible\_Single\_...

# Workshop 2—Templates versus Subsystems



This workshop takes about one half hour to complete.

## Problem statement

Understanding the difference between a template and a subsystem in ADAMS/Car is a pivotal first step toward using the full power of ADAMS/Car. To illustrate this, consider two people working side by side, both on a steering system. Looking at both computer screens, you see what appears to be the same model. However, one user is working on a template in template-builder mode, while the other is working on a subsystem in standard-interface mode. So, what's the difference?

As described before, the difference is what you can do with the models. The topology, or the way that information and parts are connected, is defined in Template Builder using parameters (variables), while defining those parameters is available in Standard Interface. Additionally, you only perform analyses in Standard Interface, based on a model (specifically, a template) created in Template Builder. A good way to understand this distinction is to create a template file and a subsystem file and compare their contents.

## Opening a template file

In this workshop, you create an ASCII template file and compare it to a subsystem file. Template files can exist either as binary or text (ASCII) files. By default, the templates saved in ADAMS/Car are binary, so to view the contents, you must save this one as text.

### To choose the template-builder mode in ADAMS/Car:

- From the **Tools** menu, select **ADAMS/Car Template Builder**.

**Note:** You can toggle between Template Builder and Standard Interface by pressing the F9 key.

## Workshop 2—Templates versus Subsystems...

---

**To open the MacPherson suspension template from the shared database:**

- 1 From the **File** menu, select **Open**.
- 2 Right-click the **Template Name** text box, point to **Search**, and then select **<shared>/templates.tbl**.
- 3 Double-click **\_macpherson.tpl**.
- 4 Select **OK**.

**To save the file as a text file:**

- 1 From the **File** menu, select **Save As**.
- 2 In the **New Template Name** text box, enter **mac\_ascii**. (Note that this text box is grayed-out because it is not a required text box to perform this function. If you do not enter a name, ADAMS/Car saves the file with its current name.)
- 3 Set **File Format** to **ASCII**. You need to do this to be able to read the file.
- 4 Select **OK**.

ADAMS/Car saves the file in the **acar\_training** database, which you had set up earlier.

# Workshop 2—Templates versus Subsystems...

## To open the file and look at the contents:

- 1 From the **Tools** menu, select **Show File** (alternatively, you can use a text editor).
- 2 Right-click the **File Name** text box, point to **Search**, and then select `<acar_training>`.
- 3 Select the directory, `templates.tbl`, and then select `_mac_ascii.tpl`.

As you can see, the file has all the information to define the model using markers, parts, communicators, forces, and so on. Take a look at the file to see what kind of information is stored. For example, the following is the beginning of the definition of the left lower control arm:

```
!  
!----- gel_lower_control_arm -----!  
!  
!  
defaults coordinate_system &  
  default_coordinate_system = ._mac_ascii.ground  
!  
part create rigid_body name_and_position &  
  part_name = ._mac_ascii.gel_lower_control_arm &  
  location = 0.0, -550.0, 150.0 &  
  orientation = 0.0d, 90.0d, 180.0d
```

# Workshop 2—Templates versus Subsystems...

## Opening a subsystem file

Subsystem files can only exist as text (ASCII) files, so you do not need to convert one from binary. You do not have to specifically create MacPherson suspension subsystem from your MacPherson template to see the major differences. Instead, you can open one based on a MacPherson suspension that already exists in the shared database.

### To open the file and look at the contents:

- 1 From the **Tools** menu, select **Show File** (alternatively, you can use a text editor).
- 2 Right-click the **File Name** text box, point to **Search**, and then select **<shared>**.
- 3 Double-click the directory, **subsystems.tbl**, and then double-click **MDI\_FRONT\_SUSPENSION.sub**.
- 4 Take a look at the file to see what kind of information it stores.

You can see that the top portion of the file looks very similar to the template file, but the rest is very different, as it resets values of parameters in the template file. Notice that in the [SUBSYSTEM\_HEADER] section, the MacPherson information is referenced for loading into your ADAMS/Car session with the line: `TEMPLATE_NAME = '<shared>/templates.tbl/_macpherson.tpl'`. Also, notice that the subsystem sets the values for the parameters in the lower control arms:

```
$-----PART_ASSEMBLY
[PART_ASSEMBLY]
  USAGE   = 'lower_control_arm'
  SYMMETRY = 'left/right'
  MASS     = 5.0911573156
$ Part location is dependent.
$ X,Y,Z location = -6.6666666667, -496.6666666667, 225.0
$ Part orientation is dependent.
$ ZP vector = -0.0652566755, -0.9951643011, -0.0734137599
$ XP vector = -0.9972332421, 0.0676631757, -0.030782389
CM_LOCATION_FROM_PART_X = 0.0
CM_LOCATION_FROM_PART_Y = 0.0
CM_LOCATION_FROM_PART_Z = 0.0
IXX      = 26908.978153
IYY      = 60577.701004
IZZ      = 33765.551131
IXY      = 0.0
IZX      = 3142.3266008533
IYZ      = 0.0
```

# Workshop 2—Templates versus Subsystems...

## Summary

Overall, a template defines the structure/topology of a model, and a subsystem redefines whatever parameters the user wants to create an instance of the template for analyses.

Below is a table that lists the characteristics of the two file types:

**Table 1: Comparison of Templates and Subsystems**

Characteristic:	Templates:	Subsystems:
Used to define structure of model	Yes	No
References the other file (template versus subsystem)	No	Yes
Can be edited to change topology (for example, the point at which force is applied)	Yes	No
Can edit parameters which define the model	Yes	Yes
Are used to perform ADAMS/Car analyses directly	No	Yes

An ADAMS/Car *template* is an ADAMS model built by an expert ADAMS/Car user in the ADAMS/Car Template Builder. The ADAMS/Car template contains geometric and topological data. The template file can be stored in ASCII or binary format.

An ADAMS/Car *subsystem* is based on an ADAMS/Car template and allows the standard user to alter the geometric data and some of the topological data of the template. The subsystem file is stored in ASCII format.

An ADAMS/Car *assembly* is a number of ADAMS/Car subsystems assembled together with an ADAMS/Car test rig. The assembly file is stored in ASCII format and is a list of the subsystems and test rig associated with the assembly

## Workshop 2—Templates versus Subsystems...

---



In this module, you learn how to create a subsystem from a template, as well as learn which parameters you can adjust in the subsystem.

### What's in this module:

- Creating Subsystems, 42
- Adjusting Hardpoints, 43
- Adjusting Parameter Variables, 44
- Adjusting Mass Properties, 45
- Adjusting Springs and Dampers, 46
- Workshop 3—Creating and Adjusting Suspensions, 48

# Creating Subsystems

---

To create a new subsystem, an existing template must be available.

Make sure you're in Standard Interface. From the File menu, point to New, and then select Subsystem. You can only create subsystems within Standard Interface.

In the New Subsystem dialog box, fill in the following text boxes:

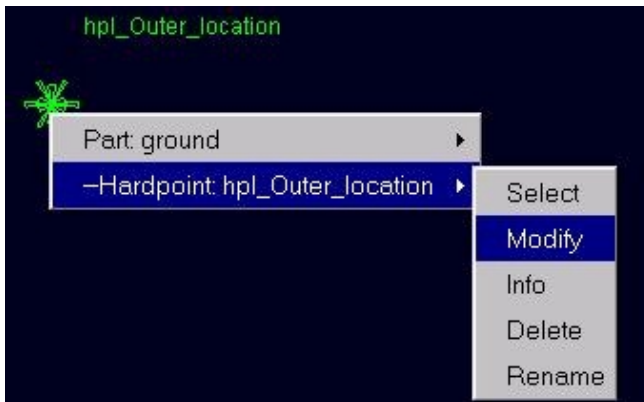
- Subsystem Name
- Minor Role
- Template Name
- Translation values (optional; lateral shifting cannot be done)

# Adjusting Hardpoints

Within a subsystem, you can move hardpoints from their default values defined in the template. Hardpoints define all key locations in your model. For more information on hardpoints, see [Creating Hardpoints, 111](#) on page 109.

In Standard Interface, from the Adjust menu, select Hardpoint. You have three options:

- **Modify** - Displays a dialog box to select one hardpoint and modify its location.
- **Table** - Displays a table with all the hardpoints in that subsystem. You can modify the location of any hardpoint in the table.
- **Info** - Displays a dialog box to select entity type and subsystem. This is already preselected to entity type of hardpoint and to the current subsystem. It will give you information about every hardpoint in the subsystem.



# Adjusting Parameter Variables

---

Within a subsystem, you can change the value of parameter variables created in Template Builder. A parameter variable is simply a variable that is used to store key information in the template. For example, in the templates, parameter variables often store the toe and camber angles for a suspension or the orientation of the spin axis. Note that parameter variables can also store text.

ADAMS/Car defines some parameter variables automatically, because they are commonly used for automotive analyses (for example, toe and camber angles). You can, however, create new parameter variables.

In Template Builder, you can create parameter variables that are hidden from standard users. Hidden parameter variables cannot be modified through Standard Interface. The naming convention for these variables is `ph[lrs]_(name):ParameterVariable_Hidden_[Left, Right, Single]`. Use hidden variables if you don't want the standard user to change particular values.

To modify parameter variables, from the **Adjust** menu, select **Parameter Variable**. You have two options:

- **Modify** - Displays a dialog box to select one parameter variable and modify its value.
- **Table** - Displays a table with all parameter variables in that subsystem, and you can modify the value of any parameter variables in the table.

# Adjusting Mass Properties

---

When the template is created, default mass properties are assigned to the bodies. You can modify these values in Standard Interface.

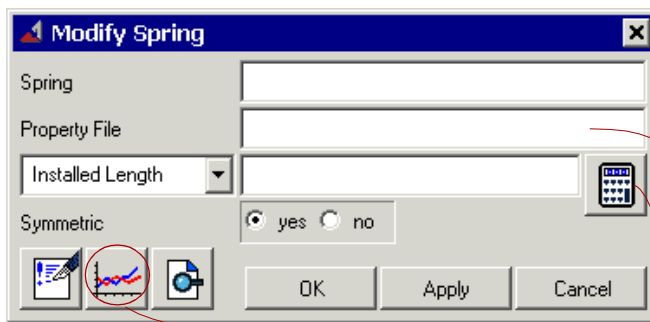
To modify mass properties, from the Adjust menu, select **General Part**. You have two options:

- **Modify** - Displays a dialog box to select a part and you can specify mass and inertia values. You can also display this dialog box by right-clicking on the part and selecting the part name followed by Modify.
- **Calculate Mass** - ADAMS/Car calculates the new values for mass and inertia based on the ADAMS/Car geometry and the density. Note that if the geometry is changed in Standard Interface from the template's default value, the respective part's mass will not automatically change. To change it, simply use the Calculate Mass function again. If your geometry is imported from a CAD package and is complex, you will have to enter the mass manually.

# Adjusting Springs and Dampers

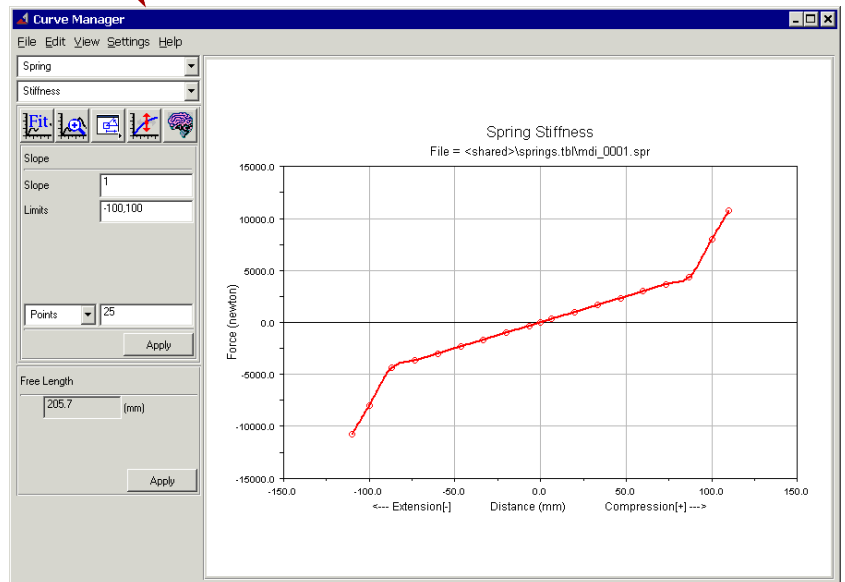
A spring or a damper is created in Template Builder and references a property file located in a particular folder in your selected database. In Standard Interface, you can link the spring or damper to a different property file or you can create a new property file.

To modify a spring, right-click the spring and select **Modify**, which displays the following dialog box:



Specify the path to the property file to be used for this spring

Calculates the required installed length for a given preload



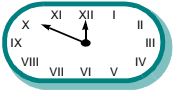
More information on the Curve Manager in subsequent chapters

## Adjusting Springs and Dampers...

---

Within the *Modify Spring* dialog box, when you right-click the property file text box, ADAMS/Car takes you to the *spring.tbl* directory in the selected database (likewise, when you right-click the property file text box in the *Modify Damper* dialog box, ADAMS/Car takes you to the *damper.tbl* directory).

# Workshop 3—Creating and Adjusting Suspensions



This workshop takes about one half hour to complete.

## Problem statement

In this workshop, you create a new subsystem and learn how to adjust its parameters.

## Creating and saving a subsystem

### To create a new subsystem:

- 1 Change to ADAMS/Car Standard Interface.
- 2 From the **File** menu, point to **New**, and then select **Subsystem**.
- 3 In **Subsystem Name** text box, enter **my\_macph**.
- 4 Set **Minor Role** to **rear**.
- 5 Right-click the **Template Name** text box, point to **Search**, and then select **<shared>templates.tbl**.
- 6 Double-click **\_macpherson.tpl**.  
ADAMS/Car informs you that the template is already in the database.
- 7 Select **Yes**.  
ADAMS/Car displays the subsystem.

### To save the subsystem:

- 1 From the **File** menu, point to **Save**, and then select **Subsystem**.

The Save Subsystem dialog box appears. Because only one subsystem is open in ADAMS/Car, by default **my\_macph** is selected. However, if you have more than one subsystem opened in your session, use the down arrow to select which one you would like to save. ADAMS/Car saves the subsystem in the database, **acar\_training**, which was set as the default database in [Setting Up Your Session](#), on page 23. ADAMS/Car saves the file in the **subsystems.tbl** table.

- 2 Select **OK**.



# Workshop 3—Creating and Adjusting Suspensions...

## Modifying a subsystem

### To open your subsystem file:

- 1 From the **Tools** menu, select **Show File**.
- 2 Right-click the **File Name** text box, point to **Search**, and then select **<acar\_training>**.
- 3 Double-click **subsystems.tbl**, and then double-click **my\_macph.sub**.

### To take a quick look at the contents of the file:

- 1 Look in the file and note the kind of information it stores. As you can see, parameters for hardpoints, spring stiffness, and so on, are defined based on the values set when created in Template Builder. To tailor this subsystem for a different MacPherson suspension, you change these values in Standard Interface, creating your own instance of the model.
- 2 For example, note the thickness of the lower arm geometry, which is 10.6823911464. You will change this value in Standard Interface, updating it in your subsystem file.

```
$-----ARM_GEOMETRY
[ARM_GEOMETRY]
USAGE      = 'lower_control_arm'
PART       = 'lower_control_arm'
SYMMETRY   = 'left/right'
THICKNESS  = 10.6823911464
```

- 3 Select **Clear**, and then close the **Info Window Read** dialog box.

### To change the lower control arm thickness:

- 1 Right-click either lower control arm (the red triangular area).
- 2 Point to **Arm: graarm\_lower\_control\_arm**, and then select **Modify**.
- 3 Change the **Thickness** to 33.3.
- 4 Select **OK**.

On your screen, note that ADAMS/Car updates the arm thickness.

### To save your subsystem:

- 1 From the **File** menu, point to **Save**, and then select **Subsystem**. Your subsystem should be in the dialog box by default.
- 2 Select **OK**, and then select **No** to prevent ADAMS/Car from creating a backup copy.

# Workshop 3—Creating and Adjusting Suspensions...

## To open and look at your file after the change:

- 1 From the **Tools** menu, select **Show File**. Your subsystem should still be entered, so select **OK**. Otherwise, right-click and search for `<private>\subsystems.tbl\my_macph.sub`.

Toward the top, you should see the arm geometry parameters updated for the thickness:

```
$-----ARM_GEOMETRY
[ARM_GEOMETRY]
USAGE      = 'lower_control_arm'
PART       = 'lower_control_arm'
SYMMETRY   = 'left/right'
THICKNESS  = 33.3
```

- 2 Close the Information window.

## To adjust hardpoints:

- 1 From the **Adjust** menu, point to **Hardpoint**, and select either of the following:

- **Modify** - Lets you adjust hardpoints one at a time
- **Table** - Lets you adjust all hardpoints.

Alternatively, you could turn icon visibility on, right-click the hardpoint you want to modify, and then select the hardpoint name followed by **Modify**.

- 2 Select **Table**.
- 3 Set **Display** to **Both**.

## To edit the size of the lower control arm:

- 1 Change **loc\_x** for either **hpl\_lca\_front** or **hpr\_lca\_front** from **-200** to **-120**. Because these hardpoints are symmetrically parameterized, changing one will automatically change the other.
- 2 Select **Apply**.
- 3 Change **loc\_x** for either **hpl\_lca\_rear** or **hpr\_lca\_rear** from **200** to **120**. Because these hardpoints are symmetrically parameterized, changing one will automatically change the other.
- 4 Select **Apply**.

The lower control arm becomes smaller.

# Workshop 3—Creating and Adjusting Suspensions...

---

## To close the hardpoint table:

- To close the table, select **OK** or **Cancel**. When you save your subsystem, ADAMS/Car updates the .sub file with these new hardpoint values.

## To adjust parameter variables:

- 1 From the **Adjust** menu, point to **Parameter Variable**, and then select **Modify**.
- 2 To see what parameter variables are available in this template, right-click the **Parameter Variable** text box, point to **Variable**, and then select **Guesses**.

Here, you can see that one parameter available for adjustment is, `pvl_toe_angle` (toe angle). Toe angle is the angle between the longitudinal axis of the vehicle and the line of intersection of the wheel plane and the road surface. ADAMS/Car reports toe angle in degrees. It is positive if the wheel front is rotated in towards the vehicle body.

- 3 Select `pvl_toe_angle`. Note that it currently has a value of 0.0.
- 4 To see the effects of toe angle, in the **Real** text box, enter **2.0**.  
ADAMS/Car updates both sides because Symmetric is set to yes.
- 5 Select **OK**.

For more related information, see the guides, *Building Templates in ADAMS/Car* and *Running Analyses in ADAMS/Car*.

## To adjust mass properties:


- 1 From the **Adjust** menu, point to **General Part**, and then select **Modify**.
- 2 Right-click the **General Part** text box, point to **General Part**, point to **Guesses**, and then select `gel_drive_shaft`.

The dialog box fills in with the relative information.

Note that only the boxes that are not grayed-out are editable. Here's the mass and inertia properties:


- Mass: 4.21745
- Ixx: 1.65989
- Iyy: 1.65989
- Izz: 692.82585

## Workshop 3—Creating and Adjusting Suspensions...

- 3 To calculate the mass based on a material in the database, select the **Calculate Mass** tool . Your selected general part is already entered.
- 4 Select a new material, such as **titanium**, and then select **OK**. Here are the new mass and inertia properties:
  - Mass: 2.39635
  - Ixx: 8.0326
  - Iyy: 8.0326
  - Izz: 399.358

### To adjust the springs:

- 1 Right-click the spring, and then select **Modify**.

Here you can adjust the property file that defines the force versus deflection, as well as either the installed length or the amount of preload, since these two quantities are directly related. The property file <shared>/springs.tbl/mdi\_0001.spr is already in the Property File text box.
- 2 To see the force-deflection curve, select the **Curve Manager** tool .

The Curve Manager replaces your Standard Interface session. Notice that the spring is a piece-wise linear spring.
- 3 To return to ADAMS/Car Standard Interface, from the **File** menu, select **Close**.
- 4 To change the property file, right-click the **Property File** text box, point to **Search**, and then select <shared>\springs.tbl\MDI\_125\_300.spr.
- 5 To see the force-deflection curve, select the **Curve Manager** tool again. Notice that this spring is a linear spring.
- 6 Return to ADAMS/Car Standard Interface.
- 7 Change the installed length to **200**.
- 8 Select **OK**.

For a spring definition, see [Springs](#), on page 138.

# Workshop 3—Creating and Adjusting Suspensions...

---

## To adjust the dampers:

This is the same procedure as for a spring.

- 1 Right-click the damper, and then select **Modify**.

Here you can adjust the property file that defines the force versus velocity. The property file <shared>/dampers.tbl/mdi\_0001.dpr is already in the Property File text box.

- 2 To see the force-velocity curve, select the **Curve Manager** tool. Notice that it is a nonlinear curve.
- 3 Return to ADAMS/Car Standard Interface.
- 4 To change the property file, right-click the **Property File** text box, point to **Search**, and then select <shared>|dampers.tbl|MDI\_default.dpr.
- 5 To see the force-velocity curve, select the **Curve Manager** tool. Notice the new nonlinear characteristics.
- 6 Return to ADAMS/Car Standard Interface.
- 7 Select **OK**.

## Reminder

All of the modifications you just made in Standard Interface only change your subsystem. There are many ways to perform these modifications. Note that the adjustments you make will change your subsystem file only after you save it.

## Workshop 3—Creating and Adjusting Suspensions...

---

# 4

## USING THE CURVE MANAGER

You use the Curve Manager to create and edit data in property files, as described in this module.

### What's in this module:

- Property File Types, 56
- Creating Property Files, 57
- Modifying an Existing Property File, 60
- Plot versus Table, 61
- Workshop 4—Modifying Springs with the Curve Manager, 62

# Property File Types

---

The Curve Manager supports the following curve types:

- Bushing
- Bumpstop
- Reboundstop
- Spring
- Damper
- Wheel envelope

The functionality of the Curve Manager changes, depending on the kind of property file being used.

The Curve Manager has two modes:

- **Plotting** - In this mode you can build a curve by specifying functions that define the curve.

For example, you can define a spring curve with a rate of 20 N/mm with 25 points between -100 and 100 mm.

- **Table** - In this mode you can specify each point in a data table.

For example, for the same spring curve made in the plotting mode, you would have to type in the x-y numbers for all 25 points.



# Creating Property Files

---

To create a new property file, in either Standard Interface or Template Builder, from the Tools menu, select Curve Manager.

To set up the Curve Manager in the appropriate mode, select a new file and specify what type of property file you want to create.

## Types of property files:

- **Bushing** - Specify all six curves that define 3-D translational and rotational stiffness. Unlike a BUSHING statement, these curves can be nonlinear.
- **Bumpstop** - A stiffness curve
- **Reboundstop** - A stiffness curve
- **Spring** - A stiffness curve and a free length (for information on springs, see [Springs](#) on page 138)
- **Damper** - A damping curve
- **Wheel envelope** - Input boundaries:
  - ◆ Steer input (length or angle)
  - ◆ Wheel interior and boundary
  - ◆ Steer interior and boundary

## Features in the Curve Manager:

- Fit the curve on the plot
- Zoom a part of the curve
- Curve math
- Vertical hotpoints
- Toggle memory curve

# Creating Property Files...

## Curve Math



- **Slope** - Specify a rate, limits, and number of points or number of segments, which is the same as number of points minus 1.
- **Offset** - Offsets the curve by the value you specify.
- **Absolute value** - No parameters, takes the absolute value of the curve.
- **Negate** - Inverts the curve.
- **Zero** - Offsets the curve so it starts from zero.
- **Y mirror** - Mirrors the y values around the middle point.
- **XY mirror** - Mirrors the x and y values, so that the curve goes through the same value for both the x and y axis.
- **Copy x->y** - Makes the y value the same as the x.
- **Function** - Specify a function, limits and number of points or segments, and you'll get a curve of the function you specified.
- **Interpolate** - Uses one of the following interpolation methods and creates the number of points you specify:

Akima	Cspline
Linear	Notaknot
Cubic	Hermite

- **Step** - Specify start and end values, and the y value for those start and end points.
- **Scale** - Scales the curve by the value you specify.
- **Ramp** - Specify start and end values, and the y value for those start and end points.
- **Expand** - Stretches the x start and end points.

## Creating Property Files...

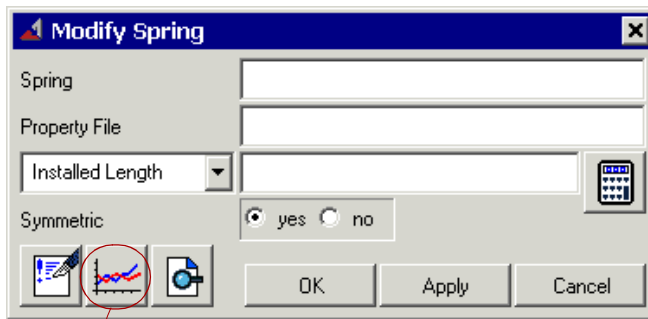
---

- **Sine** - Start and end points for x and y values, when the sweep starts, minimum and maximum amplitude, frequency and the number of points or segments.
- **Bushing** - All six curves for translational and rotational stiffness, and the damping values for each direction and for translation and rotation. You have the option of specifying a percentage value of the stiffness instead of specifying an absolute damping value.
- **Wheel envelope** - The curve math is not available.

# Modifying an Existing Property File

To modify an existing property file, you can do either of the following:

- Use the Tools menu in ADAMS/Car Standard Interface or Template Builder to open the Curve Manager, as you would to create a property file. Here, open the property file you want to edit and make your changes.
- Use a modify dialog box to open the Curve Manager, and the selected curve will automatically open for editing.



→ **Curve Manager tool**

# Plot versus Table

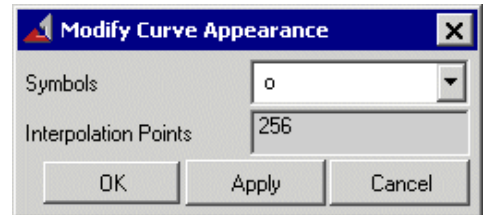
To switch between plot and table format, use the View command on the main menu and select either Plot or Table, depending on which mode you want to view. You can only close the Curve Manager, that is, return to Standard Interface or Template Builder, from the plot mode. If you are in table mode, go to plot mode and then select File -> Close.

When you finish editing the property file, you can save it. ADAMS/Car saves it to the corresponding table directory in the default writable database. For example, ADAMS/CAR saves a spring property file to the table directory spring.tbl, in the database.

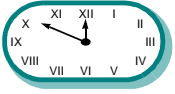
To check what the default writable database is:

- From the Tools menu, point to Database Management, and then select Database Info.
- See the subtitle in the plot, which shows the complete path to the property file.

In the Curve Manager, to change the symbols that represent the data points, from the Settings menu, select Appearance.



# Workshop 4—Modifying Springs with the Curve Manager



This workshop takes about one half hour to complete.

In this workshop, you use the Curve Manager to modify springs in a suspension analysis.



## Setting up the model

### To set up your model:

- 1 Open the **mdi\_front\_vehicle** assembly.
- 2 Run an opposite wheel travel analysis with **Output Prefix** named **baseline**.

## Modifying the spring

### To modify the spring:

- 1 Right-click the spring and select **Modify**.
- 2 Select the **Curve Manager** tool.
- 3 Right-click the **Curve Math** toolstack , and then select the **Scale** tool .
- 4 In the **Scale Value** text box, enter 1.5.
- 5 Select **Apply**.

# Workshop 4—Modifying Springs with the Curve Manager...

## Saving the spring property file

### To save the spring property file:

- 1 From the **File** menu, select **Save As**.
- 2 Name the file **my\_spring.spr**.
- 3 Select **OK**.

ADAMS/Car saves the file in your default writable database.

- 4 From the **File** menu, select **Close**.

ADAMS/Car displays a dialog box that asks if you want to reference this spring property file in your model.

- 5 Select **Yes**.
- 6 In the **Modify Spring** dialog box, select **OK**.

## Running an analysis

### To run an analysis:

- 1 Run a wheel travel analysis identical to the analysis named **baseline**, with **Output Prefix** named **new\_spring**.
- 2 In ADAMS/PostProcessor, compare the results of **dive**.

## Workshop 4—Modifying Springs with the Curve Manager...



In this module, you learn how to create a suspension assembly in ADAMS/Car. You also learn about the available suspension analyses and how to submit them.

### **What's in this module:**

- Creating Suspension Assemblies, 66
- Half-Vehicle Analyses, 67
- Suspension Parameters, 68
- Creating Loadcases, 69
- Warning Messages, 70
- Files Produced by Analyses, 71
- Workshop 5—Running Suspension Analyses, 72

# Creating Suspension Assemblies

---

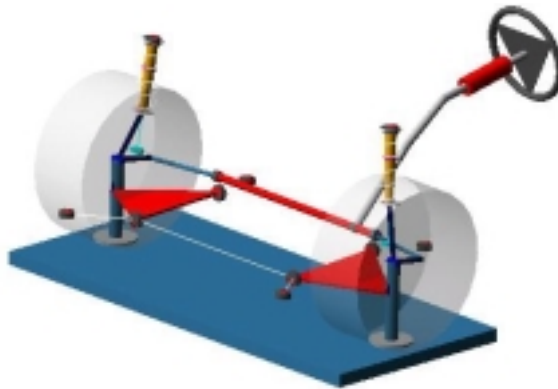
An assembly consists of a single test rig and one or more subsystems (a test rig by itself is just a specialized subsystem).

You create suspension assemblies in Standard Interface: from the File menu, point to New, and then select Suspension Assembly. In the dialog box, specify all the subsystems to be included in the assembly, as well as the test rig.

If you use subsystems created from new templates, you need to make sure the communicators match up. Beforehand, in the template-builder mode, you can test the communicators to make sure they match with other templates (communicators are described in more detail in [Communicators](#) on page 145). ADAMS/Car displays warning messages in the Message window for communicators that are not matched when creating an assembly.

The picture below shows a suspension assembly containing a suspension and steering subsystem, and test rig.

Because simulations are activated by test rigs, to perform a simulation, you must use an assembly.



# Half-Vehicle Analyses

---

You can perform the following types of suspension analyses in ADAMS/Car:

- **Parallel wheel travel** - Both wheels move up in unison.
- **Opposite wheel travel** - One wheel up, one down.
- **Single wheel travel** - One wheel fixed, while other moves.
- **Steering** - Motion applied to steering wheel or rack.
- **Static load** - Applied at specified locations (wheel center, tire contact patch).
- **External files:**
  - ◆ **Loadcase** - Essentially a selection of previous events.
  - ◆ **Wheel envelope** - A parallel wheel travel while moving the steering to get the volume the wheels take up in all exercises.

When performing a suspension analysis, ADAMS/Car uses the first second to bring the wheel centers to the lowest position, and then uses as many seconds as you specify steps, to move the suspension to the upper position.

For more information, see the guide, *Running Analyses in ADAMS/Car*.

# Suspension Parameters

---

Some of the default request outputs need information that is not available in the model. Therefore, you must supply this additional information. This information has no bearing on the outcome of the simulation, as it only affects some of the user-defined results (for example, the roll center, among others).

ADAMS/Car stores the input in an array named **Suspension Parameters**, which you can find in the **Standard Interface** under **Simulate -> Suspension Analysis -> Suspension Parameters**.

The values you must supply are:

- Loaded tire radius
- Tire stiffness
- Sprung mass
- CG height
- Wheelbase
- Drive ratio
- Brake ratio

# Creating Loadcases

---

A loadcase is an ASCII file containing all necessary information to run a simulation. It is basically a way of scripting suspension simulations with these five analysis types:

- Parallel wheel travel
- Opposite wheel travel
- Single wheel travel
- Steering
- Static load

When running a loadcase, ADAMS/Car searches for the particular loadcase file, stored in the database. You can call many loadcases, and ADAMS/Car will run them one at a time.

You can create a loadcase file by selecting Simulate -> Suspension Analysis -> Create Loadcase. Then, select the type of analysis you want to run, and specify the relevant data.

# Warning Messages

---

When you create an assembly, you will sometimes see warning messages. For example, suppose you are creating a front MacPherson suspension assembly without a steering or body subsystem. Because steering and body parts are not specified in the test rig or another subsystem, certain communicators are attached to ground or not attached to anything. The Message window displays the following:

Creating the suspension assembly: 'macph\_assy'...

Moving the rear suspension subsystem: 'my\_macph'...

Assembling subsystems...

Assigning communicators...

WARNING: The following input communicators were not assigned during assembly:

my\_macph.cil\_tierod\_to\_steering (attached to ground)

my\_macph.cir\_tierod\_to\_steering (attached to ground)

my\_macph.cis\_subframe\_to\_body (attached to ground)

my\_macph.cil\_strut\_to\_body (attached to ground)

my\_macph.cir\_strut\_to\_body (attached to ground)

my\_macph.cil\_ARB\_pickup

my\_macph.cir\_ARB\_pickup

testrig.cis\_steering\_wheel\_joint

testrig.cis\_steering\_rack\_joint

testrig.cis\_leaf\_adjustment\_steps

testrig.cis\_powertrain\_to\_body (attached to ground)

Assignment of communicators completed.

Assembly of subsystems completed.

Suspension assembly ready.

The models in the shared car database contain all the communicators that could possibly be used by other systems, and in many cases, not all communicators are used. However, be sure to check out which ones aren't being connected to see if it makes sense. Here, most of them could potentially be connected to the body, or to some other subsystem you currently don't care about in your analysis. By default, if ADAMS/Car cannot find the matching communicator, it attaches it to ground, which, in this case, is fine. So, you usually aren't concerned about it unless you see a communicator that should be used, but isn't.

# Files Produced by Analyses

---

It's important to remember that all vertical products, including ADAMS/Car, are simply preprocessors for ADAMS/Solver. This means that they simply produce the .adm file (ADAMS/Solver data, the model) and the .acf file (ADAMS/Solver commands). These files are created in the working directory (File -> Select Directory).

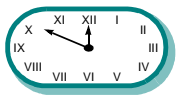
The output files produced include message, request, results, graphics, and output files. ADAMS/Car may also produce a .dcf and .dcd file, as described in [Driving Machine](#) on page 83.

When you import the analysis requests file (.req) in ADAMS/PostProcessor, ADAMS/Car also produces a special file, named the name file (.nam), which contains the name associated with every request in the interface.

You can use the .adm, .acf, and .nam files to submit an analysis outside of the ADAMS/Car graphical interface.

## Workshop 5—Running Suspension Analyses

---



This workshop takes about one hour to complete.

Go through *Suspension Analysis Tutorial* in the guide, *Getting Started Using ADAMS/Car*.



In this module, you learn how to create full-vehicle assemblies in ADAMS/Car. You also learn about the available full-vehicle analyses and how to submit them.

### What's in this module:

- Creating Full-Vehicle Assemblies, 74
- Shifting Subsystems, 75
- Updating Subsystems, 76
- Updating Assemblies, 77
- Full-Vehicle Analyses, 78
- Adjusting Mass Automatically, 80
- Workshop 6—Running Full-Vehicle Analyses, 81

# Creating Full-Vehicle Assemblies

---

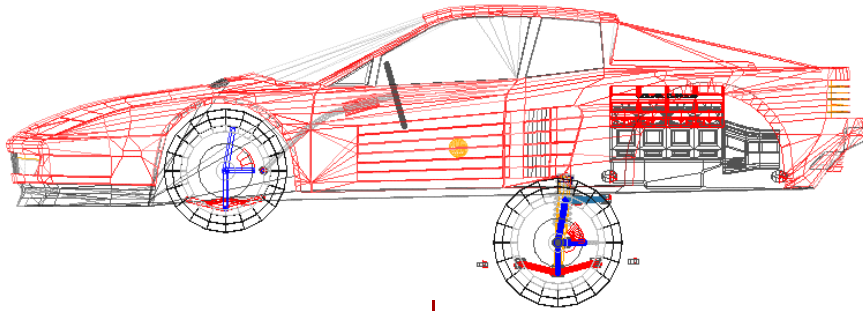
To create a full-vehicle assembly, go to File -> New -> Full Vehicle. This dialog box requires a list of all subsystems that make up your assembly. The subsystems necessary are body, front and rear suspension, front and rear tires, steering system, and a test rig.

All the analyses currently available are based on the Driving Machine. Therefore, to perform open-loop, closed-loop, and quasi-static analyses, you must select the `.__MDI_SDI_TESTRIG` in your assemblies. The only analysis that is not based on the Driving Machine is the data-driven analysis. It uses the `.__MDI_Driver_TESTRIG`.

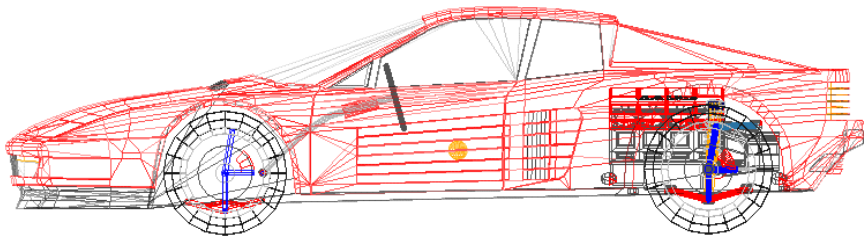
You can include other subsystems in the full-vehicle assembly by checking Other Subsystem. The location and connectivity of these subsystems depend on how the subsystems and the communicators are defined, and whether or not the template is shifted.

# Shifting Subsystems

To shift a subsystem, in the Standard Interface, go to Adjust -> Shift. Here you have the choice of shifting the subsystem fore/aft and up/down. Fore/aft moves the subsystem along the global x-axis. Up/down moves it along the global z-axis.



After shifting the rear suspension aft and up



# Updating Subsystems

---

Occasionally, you may want to change a subsystem that is used within your assembly, and see the changes take effect in your assembly. In other words, you want to edit the subsystem file, and see the changes alter your assembly file. To do this, go to File -> Update -> Subsystem, and then select the subsystem. This option prevents you from closing the assembly and reopening it with the modified subsystem to see the change.

**IMPORTANT:** Note that you must save the changed subsystem in the database, to be loaded into the assembly. Also, the update subsystem changes do not affect topology; they only update the parameters, which can be adjusted at the subsystem level.

# Updating Assemblies

---

Similarly to updating subsystems, you can update your assembly. To do so, go to File -> Update -> Assembly, and then select the assembly. This option prevents you from closing the assembly and reopening it with the modified subsystems by reloading information from the saved subsystems that the assembly references.

# Full-Vehicle Analyses

---

As stated previously, all the analyses currently available are based on the Driving Machine. Therefore, to perform open-loop, closed-loop, and quasi-static analyses, you must select the `.__MDI_SDI_TESTRIG` in your assemblies. The only analysis that is not based on the Driving Machine is the data-driven analysis. It uses the `.__MDI_Driver_TESTRIG`.

The following is a list of the types of events that are available in ADAMS/Car:

- OPEN LOOP EVENTS
  - ◆ Drift
  - ◆ Fish Hook (new to v11.0)
  - ◆ Impulse Steer
  - ◆ Ramp Steer
  - ◆ Single Lane Change
  - ◆ Step Steer
  - ◆ Swept Sine Steer
- CORNERING EVENTS
  - ◆ Breaking-In-Turn
  - ◆ Power-Off Cornering
  - ◆ Constant Radius Cornering
  - ◆ Cornering with Steering Release (new to v11.0)
  - ◆ Lift-Off Turn-In (new to v11.0)
  - ◆ Power-Off Cornering (new to v11.0)

# Full-Vehicle Analyses...

---

- STRAIGHT-LINE BEHAVIOR
  - ◆ Acceleration and Braking (Improved for v11.0)
  - ◆ Power-Off Straight-line (New to v11.0)
- COURSE EVENTS
  - ◆ ISO Lane Change
- DCF DRIVEN
- QUASI-STATIC MANEUVERS
  - ◆ Constant Radius Cornering
  - ◆ Constant Velocity Cornering
  - ◆ Force Moment Method
- DATA DRIVEN (only works with \_\_MDI\_Driver\_testrig)
- ADAMS/DRIVER

For details on these analyses, see the guide, *Running Analyses in ADAMS/Car*, or use the Dialog Box Help (F1).

On top of the dynamic events, ADAMS/Car offers a set of quasi-static events, including:

- Constant Radius Cornering
- Constant Velocity Cornering
- Force-moment method

These are available using either MDI\_DRIVER\_TESTRIG or MDI\_SDI\_TESTRIG test rig.

## Adjusting Mass Automatically

---

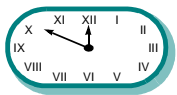
In ADAMS/Car you can adjust the mass properties of an assembled model. To adjust the aggregate mass, enter the desired mass, the desired centroidal inertias, and the desired center-of-mass location, all relative to a marker. You also select a part that ADAMS/Car modifies to match the desired mass properties. Therefore, the mass properties of the virtual vehicle match those of the real vehicle.

To adjust the mass properties, go to Simulation -> Full Vehicle Analysis -> Automatic Mass Adjustment.



## Workshop 6—Running Full-Vehicle Analyses

---



This workshop takes about one hour to complete.

Go through *Full-Vehicle Analysis Tutorial* in the guide, *Getting Started Using ADAMS/Car*.

## Workshop 6—Running Full-Vehicle Analyses...

---

The Driving Machine drives your virtual vehicle according to your instructions much like a test driver would drive an actual vehicle.

### What's in this module:

- Standard Driver Interface (SDI) and Driving Machine, 84
- Why Use SDI?, 85
- Creating Inputs for SDI, 86
- Creating .dcf and .dcd Files, 88
- Workshop 7—Editing .dcf and .dcd Files, 98

# Standard Driver Interface (SDI) and Driving Machine

---

Standard Driver Interface (SDI) is an architecture used by ADAMS/Car to perform driving analyses using inputs to drive your virtual vehicle. The manifestation of this architecture is a function in ADAMS/Car called Driving Machine. This interface controls five different channels: steering, throttle, clutch, gear, and brake. This enables you to easily recreate any physical test procedure or replay actual driving events from measured data.

Driving Machine provides a seamless interface to three vehicle controllers:

- **Open-loop control** - The open-loop controller can use either constant values or function expressions to drive the vehicle. No feedback is passed back to the controller.
- **Machine control** - The machine controller is a closed-loop controller that controls the vehicle by using the vehicle states.
- **Human control** - The human controller is, like the machine controller, a closed-loop controller, but it also has learning capabilities.

To help you calculate the control signals, the Driving Machine passes vehicle states such as position, velocity, and acceleration to your driver controller. It also provides a means for defining and passing sets of command signals, feedback signals, and parameters for each of the five control signals.

# Why Use SDI?

---

No need to create controllers or to struggle setting gains.

Shorten simulation times if conditions are achieved, or abort early if conditions are not met.

Closed-loop control for some driver signals can be combined with open-loop control for others (for example, closed-loop for steering and open-loop for throttle).

Enables users to easily recreate any physical test procedure or replay actual driving events from measured data.

Vehicle event scripting with mini-maneuvers allows easy construction of new vehicle tests.

Automatic transmission in the powertrain model.

SDI capability enables absolute control of vehicle and definition of closed-loop events by target condition (example: braking at 0.5G). So, you can build any event using the Driving Machine.

# Creating Inputs for SDI

---

Two file types are used as input for SDI-based events: driver control files (.dcf) and driver control data files (.dcd) files.

- **Driver control file (.dcf)** - Controls the event and contains a list of mini-maneuvers that allows you to script the simulation.
- **Driver control data file (.dcd)** - Contains input data used in the driver control file. This file is necessary only if it is referenced in the .dcf file.

## Driver Control File

A driver control file is a text ASCII file that you can modify in a text editor. It allows you to reference external data in a .dcd file to drive the vehicle, such as steering wheel displacement, vehicle acceleration, or other inputs. The driver control file requires four data blocks:

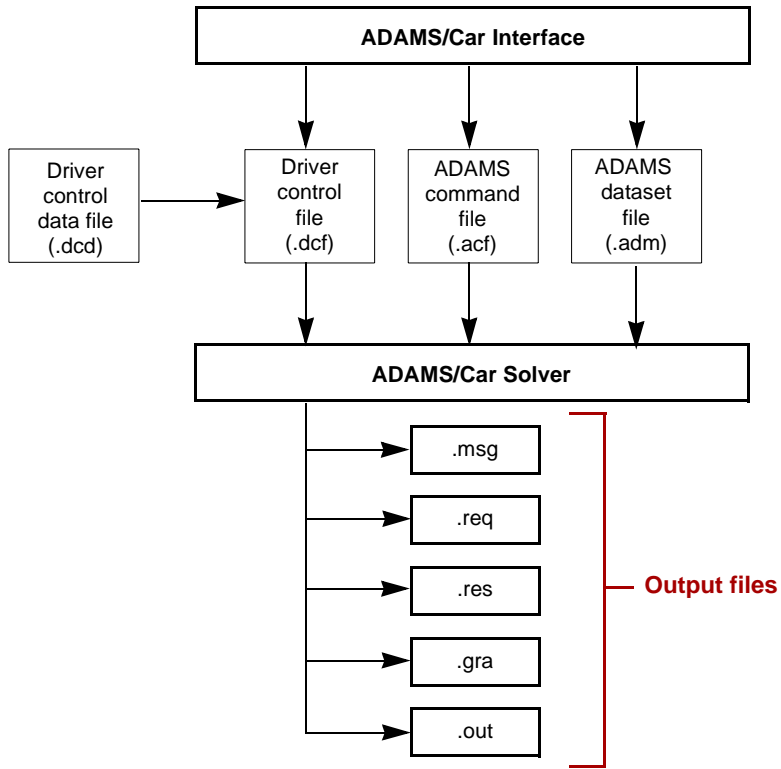
- **MDI header** - Identifies the file as a .dcf file and provides version information.
- **Units** - Sets the units of the parameters in the .dcf file. These units can be different from those used in the model.
- **Experiment** - Specifies some initial conditions for the simulation and a list of mini-maneuvers that makes up the complete experiment or event.
- **Mini-maneuver** - Specifies how the vehicle steering, throttle, brake, gear, and clutch are controlled for each maneuver, which are basically instructions for the vehicle. These names must match the names specified in the experiment block.

## Driver Control File

- **MDI header**
- **Units**
- **Controls block:**
  - ◆ **Closed-loop** - Specifies the relative data according to input parameters. For example, x, y coordinates for a vehicle path.
  - ◆ **Open-loop** - Includes data for time versus the five input channels: steer, throttle, brake, gear, and clutch.

# Creating Inputs for SDI...

The following illustration shows the Driving Machine data flow:



## Creating .dcf and .dcd Files

To create a .dcf or .dcd file you need a text editor. The preferred way is to copy a .dcf file from the shared database, and make the necessary changes to the data blocks. The following describes changes to the blocks:

- **MDI header block** - This does not require much editing: just make sure you start with a .dcf file that was created for the same version of ADAMS/Car that you intend to use.
- **Units block** - The units should be those of the data in your .dcf file, which can be different from you assembly units.
- **Experiment block** - As seen below:

```
$-----EXPERIMENT
[EXPERIMENT]
EXPERIMENT NAME = 'Constant Radius Cornering'
INITIAL_SPEED = 16.666
INITIAL_GEAR = 3
(mini_maneuver    pri_end    pri_value    abort_time    step_size
'STEADY_STATE'   'VELOCITY'    27.777       18.800        0.05
```

Here you define some initial settings and specify the list of mini-maneuvers. Note that INITIAL\_SPEED and INITIAL\_GEAR are the speed and gear, respectively, that ADAMS/Car will apply at the first time step. Last in the experiment data block is the list of the mini-maneuvers you want for the simulation. In this example, there is only one maneuver to run. The mini-maneuver names listed must match the names for the mini-maneuver block.



# Creating .dcf and .dcd Files...

For every mini-maneuver, you must specify the following:

- **Pri\_end** - Primary end condition, which is the state you want to control the termination. This could be one of the following:
  - ◆ TIME
  - ◆ DISTANCE
  - ◆ VELOCITY
  - ◆ ACCELERATION
- **Pri\_value** - The value to achieve for the primary end condition, +/- 0.2%.
- **Abort\_time** - The time when that mini-maneuver will be terminated, regardless if the primary end condition has been achieved or not.
- **Step\_size** - The solver output step size.

(Note that new to v11.0, multiple end conditions can be specified, when the .dcf File Version is 2.0. These end conditions can be for time, distance, velocity, acceleration (longitudinal and lateral), and yaw (acceleration, displacement, and velocity). For details, see the guide, *Running Analyses in ADAMS/Car*.

For every mini-maneuver you specify in the experiment block, you must specify a separate mini-maneuver sub-block, that might look as shown next:

```
$-----STEADY_STATE
[STEADY_STATE]
(STEERING) ← this is the name of the mini-maneuver;
               it must match the name in the
               experiment block
    ACTUATOR_TYPE    = 'ROTATION'
    METHOD            = 'MACHINE'
(THROTTLE)
    METHOD            = 'MACHINE'
(BREAKING)
    METHOD            = 'MACHINE'
```

## Creating .dcf and .dcd Files...

---

```
(GEAR)
  METHOD          = 'OPEN'
  MODE           = 'ABSOLUTE'
  CONTROL_TYPE   = 'CONSTANT'
  CONTROL_VALUE  = 3
(CLUTCH)
  METHOD          = 'OPEN'
  MODE           = 'ABSOLUTE'
  CONTROL_TYPE   = 'CONSTANT'
  CONTROL_VALUE  = 0
(MACHINE_CONTROL)
  STEERING_CONTROL = 'SKIDPAD'
  RADIUS           = 80.0
  TURN_ENTRY_DISTANCE = 30.0
  TURN_DIRECTION  = 'LEFT'
  SPEED_CONTROL   = 'VEL_POLYNOMIAL'
  VELOCITY        = 16.666
  ACCELERATION    = .855
  JERK            = 0.0
  START_TIME      = 3.800
```

Here, the mini-maneuver name (STEADY\_STATE) is specified between the first set of square brackets, and must match the name listed in the experiment block. Every mini-maneuver sub-block requires five attributes: steering, throttle, braking, gear, and clutch, which are specified with parentheses (Here, (MACHINE\_CONTROL) is an extra block to perform this function).

# Creating .dcf and .dcd Files...

---

Within the steering, throttle, braking, gear, and clutch, various parameters are assigned. Notes on these parameters are listed next:

## ACTUATOR\_TYPE

Allows choice of actuator input.

- **ROTATION** - Applies an angular motion to the steering column joint.
- **TRANS** - Applies a translational motion to the steering rack joint.
- **FORCE** - Applies a force to the steering rack.
- **TORQUE** - Applies a torque to the steering wheel.

## METHOD

Specifies the control method to use.

- **OPEN\_CONTROL** - Must be defined as a function of time. Must define the CONTROL\_TYPE argument.
- **MACHINE\_CONTROL** - Uses the Driving Machine controllers to drive the vehicle.
- **HUMAN\_CONTROL** - Uses ADAMS/Driver to control the vehicle.

## MODE

Specifies whether to carry signals on from previous maneuvers or set a new value.

- **RELATIVE** - Maintains the last signal from the previous mini-maneuver as the initial input for the next mini-maneuver.
- **ABSOLUTE** - Forces the initial input to be the value specified, which can cause discontinuities in signals.

## CONTROL\_TYPE

For steering, throttle, brake, clutch, and gear, you can define the control\_type as:

- ◆ Constant
- ◆ Step
- ◆ Ramp
- ◆ Impulse
- ◆ Sine
- ◆ Swept\_sine
- ◆ Data\_driven

# Creating .dcf and .dcd Files...

---

For every control type **MACHINE CONTROL**, you must include a separate data block.

## STEERING

- **FILE** - Supply a .dcd file.
- **STRAIGHT** - Controls the vehicle to go straight from the initial position.
- **SKIDPAD** - Controls the vehicle to drive on a skidpad. The following parameters need to be defined as well.
- **TURN\_DIRECTION** - Left or right.
- **TURN\_ENTRY\_DISTANCE** - Distance from start to turn.
- **RADIUS** - Radius of skidpad.
- **MAINTAIN** - Maintains the initial speed of the vehicle. The Driving Machine will control the throttle to maintain the initial speed.

## SPEED\_CONTROL

- **FILE** - Supply a .dcd file.
- **LAT\_ACCEL** - Controls the lateral acceleration.
- **LAT\_ACCEL\_TARGET** - Lateral acceleration target.
- **MAX\_ENGINE\_SPEED** - When it's time to change gear.
- **MIN\_ENGINE\_SPEED** - When it's time to shift down.

## VEL\_POLYNOMIAL

Defines a velocity curve.

- **VELOCITY** - Velocity
- **ACCELERATION** - Acceleration
- **JERK** - Square of acceleration
- **START\_TIME** - The time when we hit the throttle.
- The following equation is used:

IF (Time < START\_TIME ):

SPEED = VELOCITY

IF (Time > START\_TIME ):

SPEED = VELOCITY + ACCELERATION\*(TIME – START\_TIME)+1/2\*  
JERK(TIME-START\_TIME)\*\*2

# Creating .dcf and .dcd Files...

---

For HUMAN control, ADAMS/Driver is required with a separate license. You cannot switch from an open-loop controlled mini-maneuver to a human mini-maneuver, but you can switch from human to open. ADAMS/Driver approximates the behavior of a human driver and is capable of learning and adapting to the characteristics of the vehicle.

## Arguments used when using HUMAN control:

- DRIVER\_INPUT\_FILE = STRING <filename.din> - This file specifies the option that ADAMS/Driver uses to control the vehicle.
- DRIVER\_ROAD\_FILE = STRING <filename.drd> - This file specifies the road as a set of x, y coordinates and lane widths.
- START\_DRIVER\_ACTIVITIES = VALUE <time> - Enter the time after the beginning of the mini-maneuver when you want ADAMS/Driver to start controlling the vehicle. If the mini-maneuver is the first in the experiment (simulation), this time must be greater than zero.
- LEARNING\_ACTIVITIES = 'LATERAL\_DYNAMICS' || 'LONGITUDINAL\_DYNAMICS' || 'BASIC\_DYNAMICS' || 'LIMIT\_HANDLING' || 'NONE'

# Creating .dcf and .dcd Files...

ADAMS/Driver has the ability to learn and adapt to a particular vehicle's characteristics. Using **LEARNING\_ACTIVITIES** you specify the kind of learning ADAMS/Driver does. ADAMS/Driver stores what it learns about your vehicle in the output knowledge file for use in subsequent simulations.

- **LATERAL\_DYNAMICS** - Learns and adapts to the vehicle's lateral dynamics only.
- **LONGITUDINAL\_DYNAMICS** - Learns and adapts to the vehicle's longitudinal dynamics only.
- **BASIC\_DYNAMICS** - Learns and adapts to both the vehicle's lateral and longitudinal dynamics.
- **LIMIT\_HANDLING** - ADAMS/Driver learns and adapts to the vehicle's limit handling characteristics. To learn the limit handling characteristics of a vehicle, ADAMS/Driver attempts to drive the vehicle as quickly as possible.
- **NONE** - ADAMS/Car does not record any of the information resulting from the mini-maneuver.

## REMEMBER = 'YES' || ;NO'

Select an option for REMEMBER. When you select YES, ADAMS/Driver reads the knowledge file you specify using the KNOWL\_INPUT\_FILE argument. Each time you run ADAMS/Driver, it creates a knowledge file, KNOWL\_OUTPUT\_FILE, to store what it learns about the vehicle's characteristics.

**For every control type HUMAN CONTROL, you must include a separate data block:**

```
(HUMAN_CONTROL)
DRIVER_INPUT_FILE      = '<shared>/mdi_driver_001.din'
DRIVER_ROAD_FILE       = '<shared>/MDI_track.drd'
START_DRIVER_ACTIVITIES= 0.5
LEARNING_ACTIVITIES    = 'LIMIT_HANDLING'
REMEMBER               = 'YES'
KNOWL_INPUT_FILE       = 'limit_handling.kno'
KNOWL_OUTPUT_FILE      = 'limit_handling1.kno'
INITIAL_GEAR           = 3
```



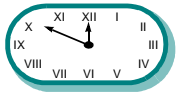
## Creating .dcf and .dcd Files...

The following table summarizes the closed-loop data that a .dcd file may contain. The columns represent speed-control options from the driver parameters array. The rows represent the steering control options from the driver parameters array. The intersections give the data contained in the .dcd file and, thus, the data input to the funnel to produce {x,y,vt} as needed by Driver-Lite.

SPEED_CONTROL STEERING_CONTROL	none	lon_vel (p1=0)	lon_acc (p1=1)	lat_acc (p1=2)	path (p1=3)
none	NOT VALID	{(distance or time), lon_vel}	{(distance or time), lon_acc}	NOT VALID	NOT VALID
curvature (p1 = 0)	{distance, curvature}	{(distance or time), curvature, lon_vel}	{(distance or time), curvature, lon_acc}	{(distance or time),curvature, lat_acc}	NOT VALID
path (p1 = 1)	{x, y}	{x, y, vt}	{x, y, lon_acc}	{x, y, lat_acc}	{x, y, time}
lat_acc (p1 = 2)	NOT VALID	{distance or time, lat_acc, lon_vel}	{distance or time, lat_acc, lon_acc}	NOT VALID	NOT VALID

For example, if STEERING\_CONTROL = 'path', and SPEED\_CONTROL = 'none', then in the (DATA) block of your .dcd file, you need a block of {X Y} data to define the path of your vehicle. (The p1 parameters are the values passed to the SDI arrays in the .adm file.)

# Workshop 7—Editing .dcf and .dcd Files



This workshop takes about one hour to complete.

## Problem statement

In this workshop, you edit a .dcf and .dcd file from existing files in the default ADAMS/Car shared database.

## Example .dcf file

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_NAME   = iso_lane_change.dcf
FILE_TYPE   = 'dcf'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'Example DCF file for Closed Loop ISO-Lane Change'

$-----UNITS
[UNITS]
LENGTH = 'meters'
FORCE   = 'newton'
ANGLE   = 'radians'
MASS    = 'kg'
TIME    = 'sec'

$-----EXPERIMENT
[EXPERIMENT]
EXPERIMENT_NAME = 'ISO-Lane Change'
INITIAL_VELOCITY = 16.667
INITIAL_GEAR     = 3
{mini_manuever   pri_end   pri_value   abort_time   step_size}
'LANE_CHANGE'    'TIME'    12.0       12.0        0.05
```

## Workshop 7—Editing .dcf and .dcd Files...

```
$-----LANE_CHANGE
[LANE_CHANGE]
(STEERING)
  ACTUATOR_TYPE    = 'ROTATION'
  METHOD            = 'MACHINE'
(THROTTLE)
  METHOD            = 'MACHINE'
(BRAKING)
  ACTUATOR_TYPE    = 'FORCE'
  METHOD            = 'MACHINE'
(GEAR)
  METHOD            = 'OPEN'
  MODE             = 'ABSOLUTE'
  CONTROL_TYPE     = 'CONSTANT'
  CONTROL_VALUE    = 3
(CLUTCH)
  METHOD            = 'OPEN'
  MODE             = 'ABSOLUTE'
  CONTROL_TYPE     = 'CONSTANT'
  CONTROL_VALUE    = 0
(MACHINE_CONTROL)
  STEERING_CONTROL = 'FILE'
  DCD_FILE         = 'iso_lane_change.dcd'
  SPEED_CONTROL    = 'VEL_POLYNOMIAL'
  VELOCITY         = 16.667
  ACCELERATION     = 0.0
  JERK             = 0.0
  TRANSITION_TIME  = 0.0
  MIN_ENGINE_SPEED = 750
  MAX_ENGINE_SPEED = 6500
```

Here is the supporting .dcd file for the ISO lane change:

```
[MDI_HEADER]
FILE_NAME   = iso_lane_change.dcd
FILE_TYPE   = 'dcd'
FILE_VERSION = 1.0
FILE_FORMAT = 'ASCII'
(COMMENTS)
{comment_string}
'Example DCD file of ISO-Lane Change Path'
```

# Workshop 7—Editing .dcf and .dcd Files...

```
$-----UNITS
[UNITS]
LENGTH = 'meters'
FORCE = 'newton'
ANGLE = 'radians'
MASS = 'kg'
TIME = 'sec'

$-----CLOSED_LOOP
[CLOSED_LOOP]
STEERING_CONTROL = 'path'
SPEED_CONTROL = 'none'

(DATA)
{ X   Y }
 0.0 0.000
45.0 0.000
52.5 0.000
60.0 0.000
90.0 3.5
102.0 3.5
115.0 3.5
140.0 0
147.0 0
155.0 0
162.0 0
170.0 0
200.0 0
300.0 0
400.0 0
500.0 0
```

## Working with the default .dcf file

In this section, you open an assembly, perform a full-vehicle analysis on the default .dcf file, and then review the results.

### To open an assembly:

- 1 From the **File** menu, point to **Open**, and then select **Assembly**.
- 2 Search the shared database for **MDI\_Demo\_Vehicle.asy**.

# Workshop 7—Editing .dcf and .dcd Files...

---

## To perform the full-vehicle analysis:

- 1 From the **Simulate** menu, point to **Full-Vehicle Analysis**, and then select **DCF Driven**.
- 2 In the **Output Prefix** text box, enter **default**.
- 3 Right-click the **Driver Control Files** text box, and from <shared>\driver\_controls.tbl, double-click **iso\_lane\_change.dcf**.
- 4 Select **OK**.

## To review results:

- 1 From the **Review** menu, select **Postprocessing Window** or press the **F8** key.
- 2 From the **Simulation** list, select **default\_iso\_lane\_change**.
- 3 From the **Filter** list, select **user\_defined**.
- 4 From the **Request** list, select **chassis\_displacements**.
- 5 From the **Component** list, select **lateral**.
- 6 Select **Add Curves**.

The plot displays the lateral displacement of the chassis, as defined in your driver control data file. The plot shows how the car drives from one lane to another and back.

## To copy the .dcf and .dcd files into your private database:

- 1 From the **Tools** menu, point to **Database Management**, and then select **Database Info** to check the path to the shared database.
- 2 From the shared database, copy the ISO lane change .dcf file, **iso\_lane\_change.dcf**, into your private database.
- 3 Rename it to **iso\_lane\_change\_mod7.dcf**.
- 4 From the shared database, copy the .dcd file, **iso\_lane\_change.dcd**, into your private database.
- 5 Rename it **iso\_lane\_change\_mod7.dcd**.

# Workshop 7—Editing .dcf and .dcd Files...

## Editing the .dcd and .dcf files

Modify the .dcd file by increasing the displacement of the lane change, and then change the names in your .dcf and .dcd files to correspond to their new names. For the .dcd file, rename the file name in the MDI\_HEADER block. For the .dcf file, rename the file name in the MDI\_HEADER and the LANE\_CHANGE mini-maneuver blocks.

### To edit the iso\_lane\_change\_mod7.dcd file:

- Change the three entries in the Y column data from 3.5 to 10, as seen in this excerpt:

```
$-----CLOSED_LOOP
[CLOSED_LOOP]
STEERING_CONTROL = 'path'
SPEED_CONTROL = 'none'

(DATA)
{ X      Y }
  0.0    0.000
  45.0    0.000
  52.5    0.000
  60.0    0.000
  90.0    10
 102.0    10
 115.0    10
 140.0    0
...
```

## Running an analysis and reviewing its results

Run a full-vehicle analysis on the modified .dcf file in your private database.

### To perform the full-vehicle analysis:

- 1 From the **Simulate** menu, point to **Full-Vehicle Analysis**, and then select **DCF Driven**.
- 2 In the **Output Prefix** text box, enter **mod7\_iso**.
- 3 Right-click the **Driver Control File** text box, point to **Search**, and then select **<private>\driver\_controls.tbl**.
- 4 Double-click **iso\_lane\_change\_mod7.dcf**.
- 5 Select **OK**.

# Workshop 7—Editing .dcf and .dcd Files...

## To review the analysis results:

- 1 Launch ADAMS/PostProcessor.  
You should still see the plot from the first lane change analysis.
- 2 From the **Simulation** list, select **mod7\_iso**.
- 3 From the **Filter** list, select **user\_defined**.
- 4 From the **Request** list, select **chassis\_displacements**.
- 5 From the **Component** list, select **lateral**.
- 6 Select **Add Curves**.

The plot shows that the car has traveled farther than in the first analysis. It does not exactly meet the desired path of 10 m (10000 mm), because a closed-loop controller performs the maneuver to provide the best possible response given the desired output and simulation conditions. If, say, we had given the Driving Machine more time and/or greater distance to perform this maneuver, it would be able to more closely meet the 10 m desired path.

So, for example, if you changed your .dcd file as shown next, you would see that the lateral displacement would be closer to the desired value of 10 m:

```
(DATA)
{ X      Y }
  0.0    0.000
 45.0    0.000
 52.5    0.000
 60.0     2
 90.0     4
102.0     6
115.0     8
140.0    10
147.0    10
155.0     8
162.0     6
170.0     4
200.0     2
300.0    0.172
400.0    0.172
500.0    0.172
```

For more tips on the Driving Machine, see the knowledge base article 9058 (key words: ADAMS/Driver, ADAMS/Driver-Lite, ADAMS Driving Machine, SDI, test rigs) at: <http://support.adams.com/kb/faq.asp?ID=kb9058.html>.

## Workshop 7—Editing .dcf and .dcd Files...

---



This module explains plot configuration files, how to create them, and how to review the results of your analyses.

**What's in this module:**

- [What is a Plot Configuration File?, 106](#)
- [Workshop 8—Creating Plot Configuration Files, 107](#)

# What is a Plot Configuration File?

---

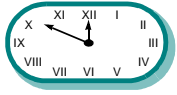
A plot configuration file defines a series of predefined plots created by using analysis results. This file is a collection of plot templates that stores the information about what output components to use for vertical and horizontal axes, titles, labels, scaling, legends, and so on. These files set up the commands to create the plots, not the actual plots. This file prevents you from creating each individual plot every time you run a particular simulation. This file is stored in the `plot_configs` directory (table) as an ASCII file.

To create a plot configuration file, first create the desired plots in ADAMS/PostProcessor (pages and curves). Then, go to `Files -> Export -> Plot Config`.

To load a plot configuration file, while in ADAMS/PostProcessor, go to `Plot -> Create Plots`, and specify the analysis and the plot configuration file (\*.plt).

You can also include a set of ADAMS/View commands to execute after you've loaded the defined plots. These commands can change the format of the plots or modify the data associated with a curve.

# Workshop 8—Creating Plot Configuration Files



This workshop takes about one half hour to complete.

## Problem statement

For this workshop, simply create a series of suspension plots and save a plot configuration file.

## Basic plot configuration steps:

- 1 Run a suspension analysis in the Standard Interface.
- 2 Make plots by hand in ADAMS/PostProcessor.
- 3 Save the plot configuration file (.plt) in ADAMS/PostProcessor.
- 4 Re-run the analysis in Standard Interface.
- 5 Load the plot configuration file (.plt) in ADAMS/PostProcessor.

The same plots are made automatically with the plot configuration file.

## Workshop 8—Creating Plot Configuration Files...

---

This module explains how ADAMS/Car models are parameterized, by location and orientation, using hardpoints and construction frames.

### What's in this module:

- Parameterization in ADAMS/Car, 110
- Creating Hardpoints, 111
- Creating Construction Frames, 113
- Location Parameterization, 115
- Orientation Parameterization, 120

## Why parameterize?

Parameterizing a template allows you to build relationships into the model so that when you change a modeling entity, ADAMS/Car automatically updates all other entities that depend on it. You can, therefore, build a whole vehicle model to depend on only a few key hardpoints and variables, saving time and effort in making design changes.

## What can you parameterize?

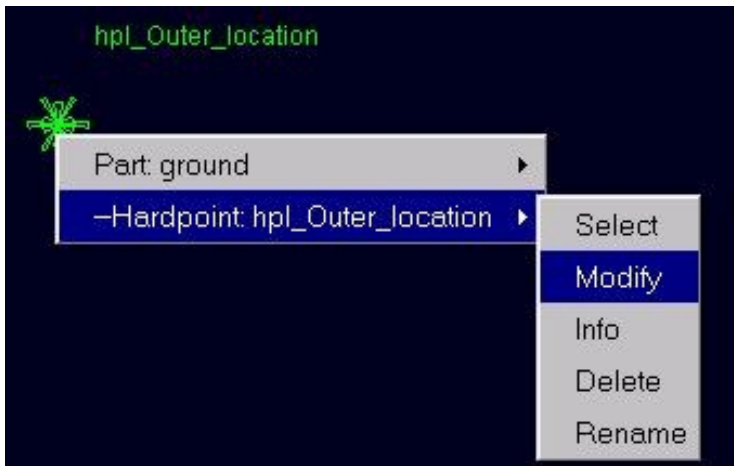
- Location and orientation expressions
- Geometry
- Group activity
- Functions
- And so on

# Creating Hardpoints

Hardpoints define all key locations in your model. They are the most elementary building blocks that you use to parameterize locations for higher-level entities, such as construction frames, parts, and attachments. Hardpoints are the same as points in ADAMS/View.

You create hardpoints in Template Builder. To create hardpoints, go to Build -> Hardpoints -> New. In the dialog box, specify the name of the hardpoint, if it's a left, right or single, and the location. When you create a left or right hardpoint, ADAMS/Car creates a corresponding paired hardpoint by reflecting the location along the car's longitudinal axis.

You can modify hardpoints by right-clicking the hardpoint and selecting the hardpoint name followed by Modify.



## Creating Hardpoints...

---

You can also modify hardpoints by going to **Build -> Hardpoint**, where you can select:

- **Table** - Every hardpoint in the model appears and you can enter new locations for all.
- **Modify** - You can modify only one hardpoint at a time.



# Creating Construction Frames

---

Construction frames are building blocks that you use whenever an entity requires that you specify an orientation in addition to a location. Construction frames are the same as markers in ADAMS/View.

To create construction frames, go to Build -> Construction Frame -> New. In the dialog box, specify the name and if it's a left, right, or single. Define the location and orientation, by selecting one of the following options.

## Location parameterization

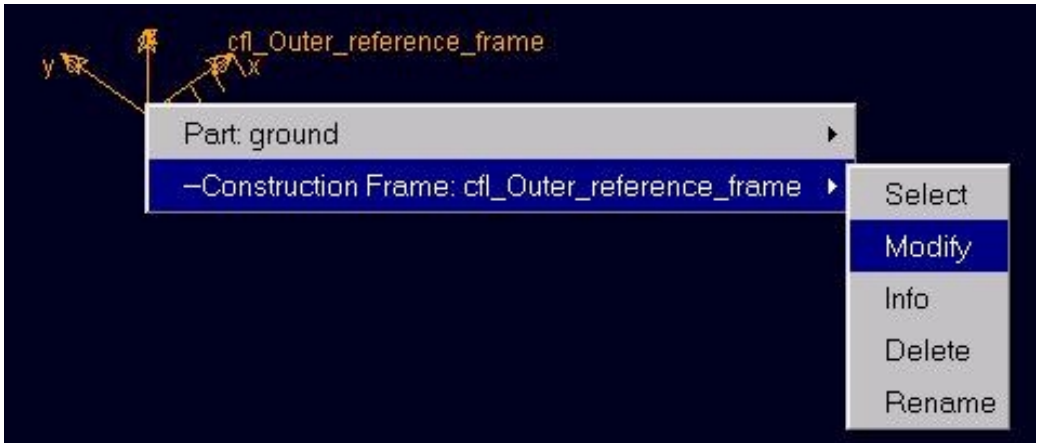
- Delta location from coordinate
- Centered between coordinates
- Located on a line
- Located along an axis
- Location input communicator
- Located at flexible body node

## Orientation parameterization

- User entered values (Euler angles)
- Delta orientation from coordinate
- Parallel to axis
- Oriented in plane
- Orient to zpoint - xpoint
- Orient axis along line
- Orient axis to point
- Orientation input communicator
- Toe/camber

## Creating Construction Frames...

To modify construction frames, right-click the construction frame and select the construction frame name followed by **Modify**.

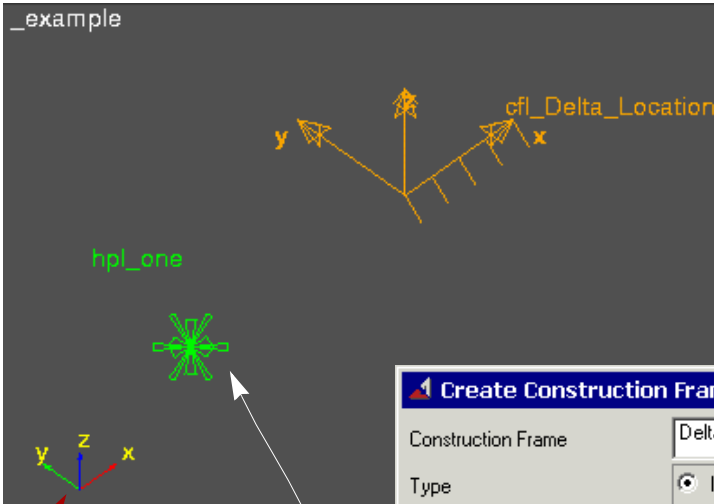


You can also modify construction frames by going to **Build -> Construction Frame -> Modify**. In the dialog box, select the name of the construction frame you want to modify. You can modify the location and orientation.

# Location Parameterization

## Delta location from coordinate

- Locate with respect to a defined reference frame.
- You can define X, Y, Z displacement in the local or global coordinate reference frame.



Global reference frame

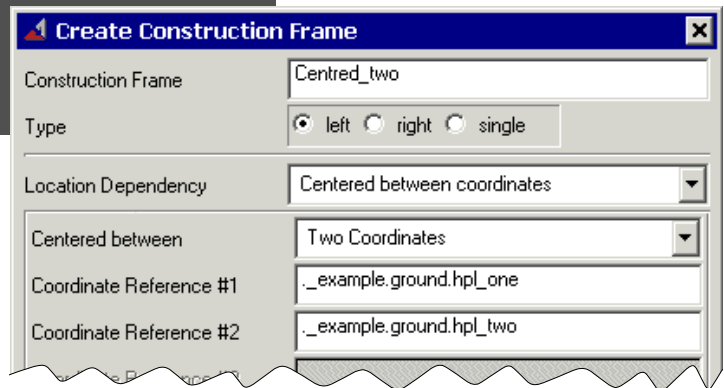
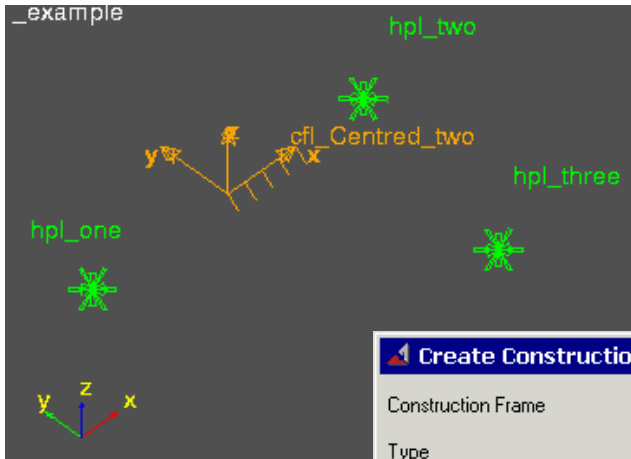
Coordinate reference

Location is 100 length units along global x-axis

# Location Parameterization...

## Centered between coordinates

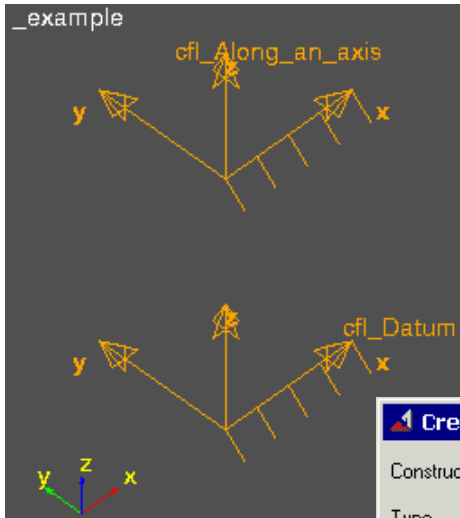
- Using the two-coordinates method, the entity is located on the mid-point along an imaginary line joining the defined reference coordinates.
- Using the three-coordinate method, the entity is located on the center point of a plane defined by the three reference coordinates.



# Location Parameterization...

## Located along an axis

- The entity is located a defined distance along the chosen axis of the reference construction frame.
- In this example, the entity has been located 100 length units along the z-axis of the reference construction frame.



**Create Construction Frame**

Construction Frame:

Type: ☒ left ☐ right ☐ single

Location Dependency:

Construction Frame:

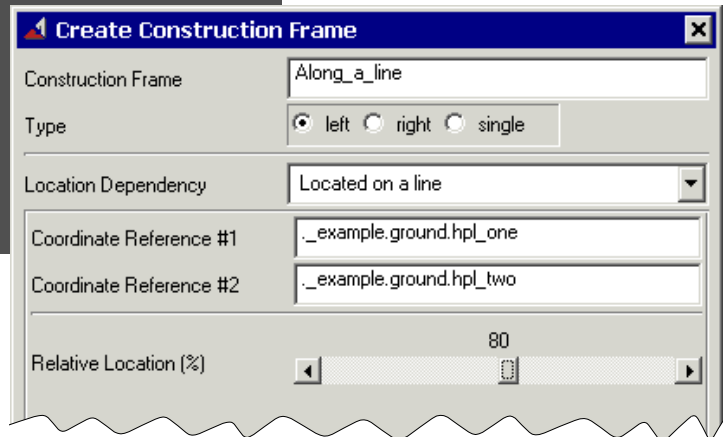
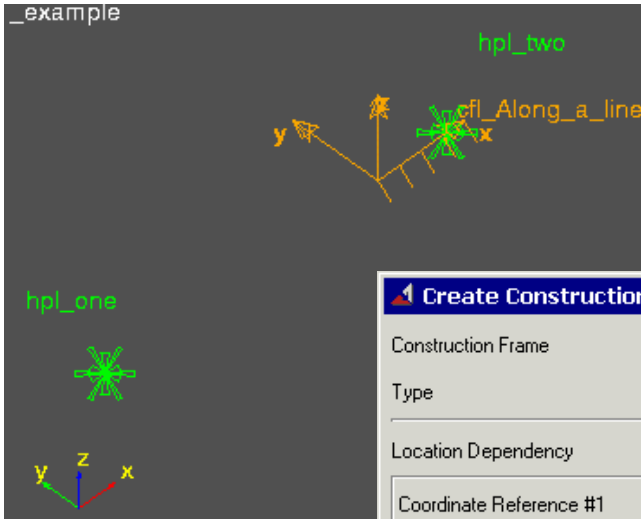
Distance:

Axis: ☐ X ☐ Y ☒ Z

# Location Parameterization...

## Located on a line

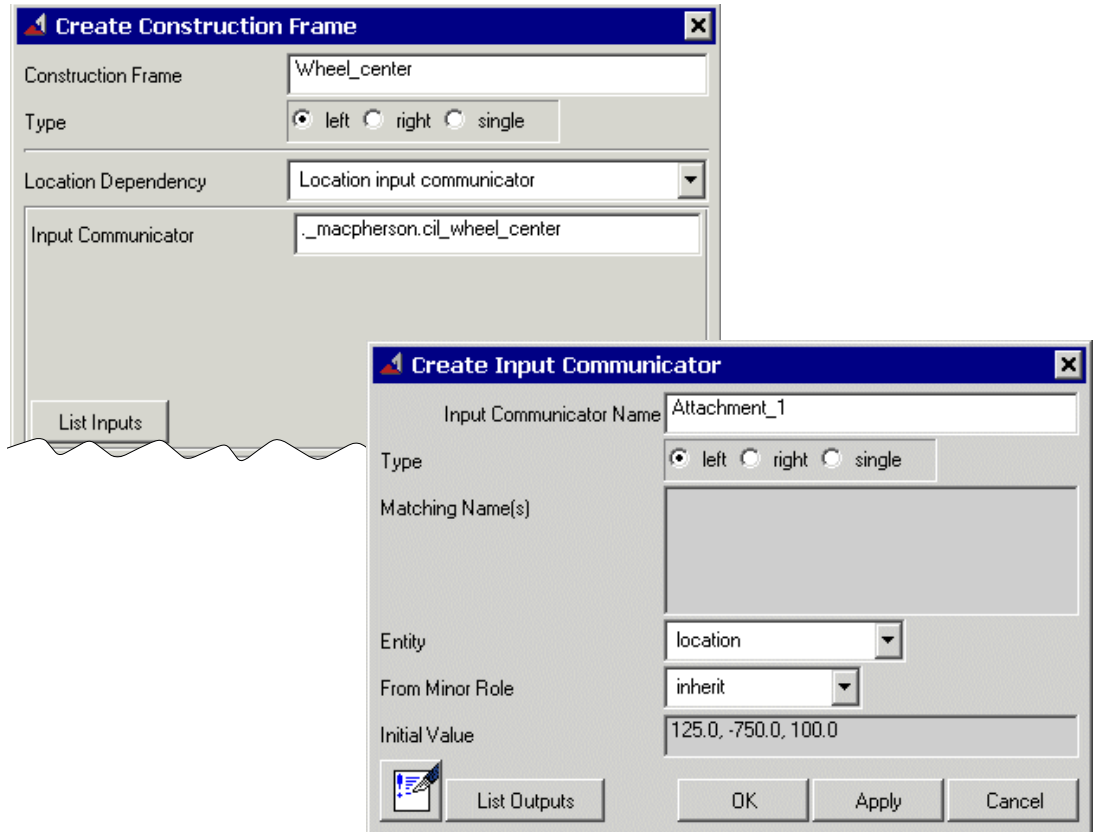
- Locates the entity along a line defined by the two reference coordinates.
- The entity is located a defined percentage along the line measured from the first reference coordinate to the second reference coordinate.



# Location Parameterization...

## Location input communicator

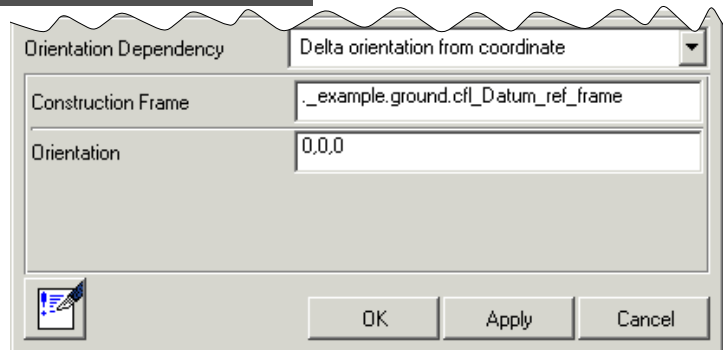
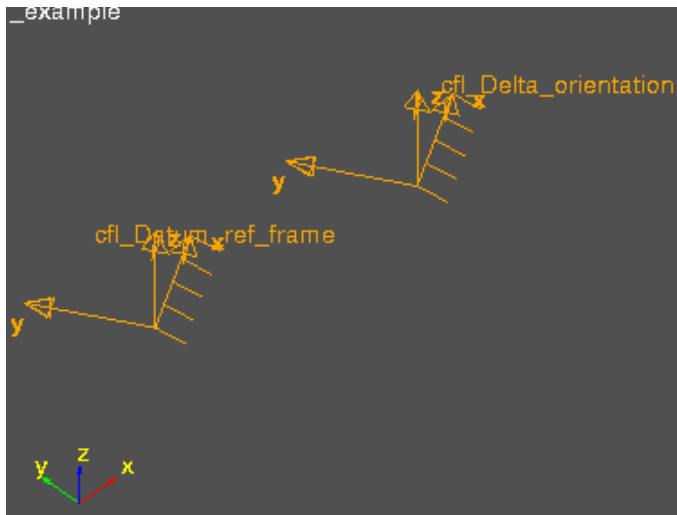
- Locates the entity using location data from the chosen input communicator.
- Use this option to locate entities with respect to reference frames in other templates.
- At the assembly stage, the location data is communicated to the input communicator from the corresponding output communicator in the other template. Up to that point, the entity is located at the initial value defined in the input communicator.



# Orientation Parameterization

## Delta orientation from coordinate

- Orients the entity with respect to the reference frame, using the Euler angle orientation defined.
- In this example, the entity has been defined such that there is no change in orientation with respect to the reference construction frame.

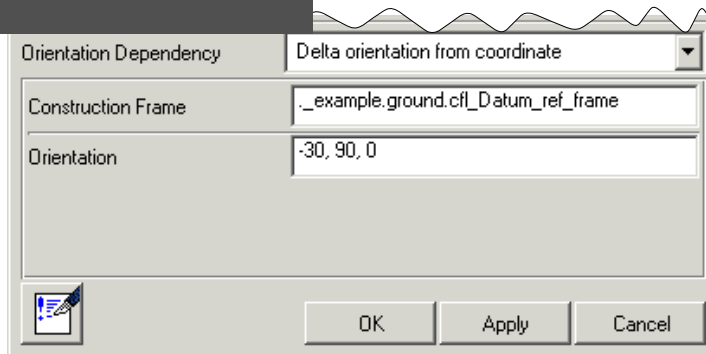
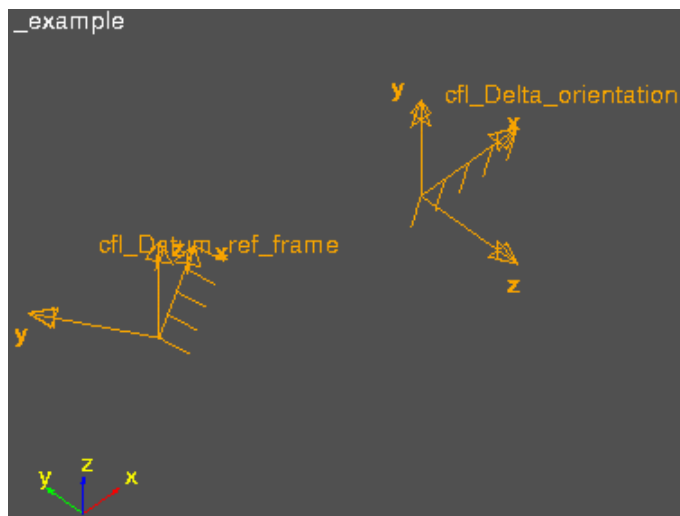




# Orientation Parameterization...

## Delta orientation from coordinate

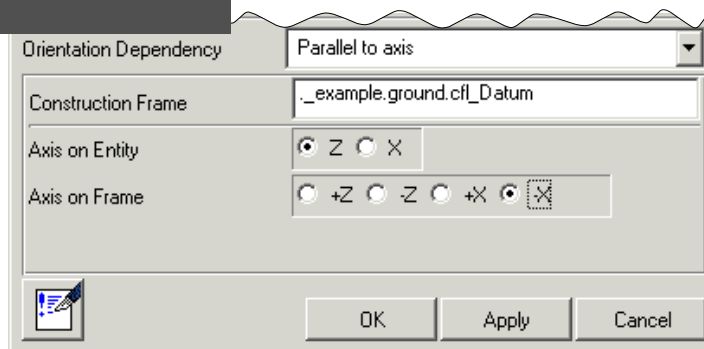
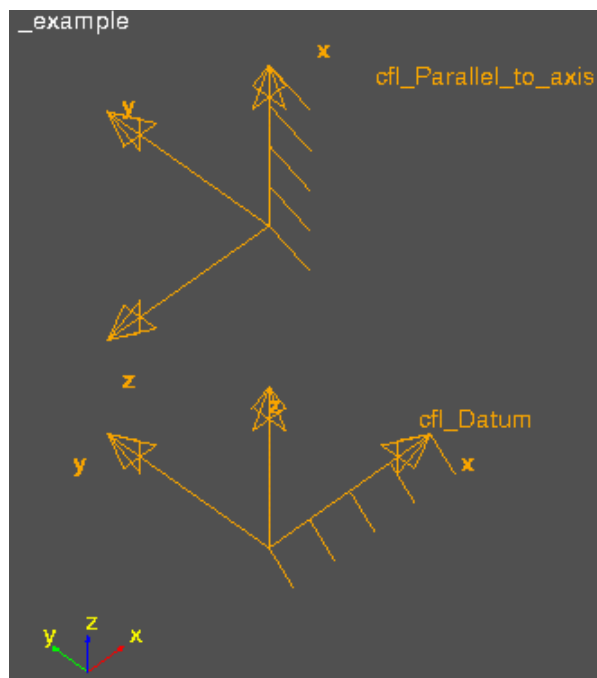
- Orients the entity with respect to the reference frame, using the Euler angle orientation defined.
- In this example, the entity has been rotated -30 degrees about the reference frame z-axis and then 90 degrees about the new x-axis.



# Orientation Parameterization...

## Parallel to axis

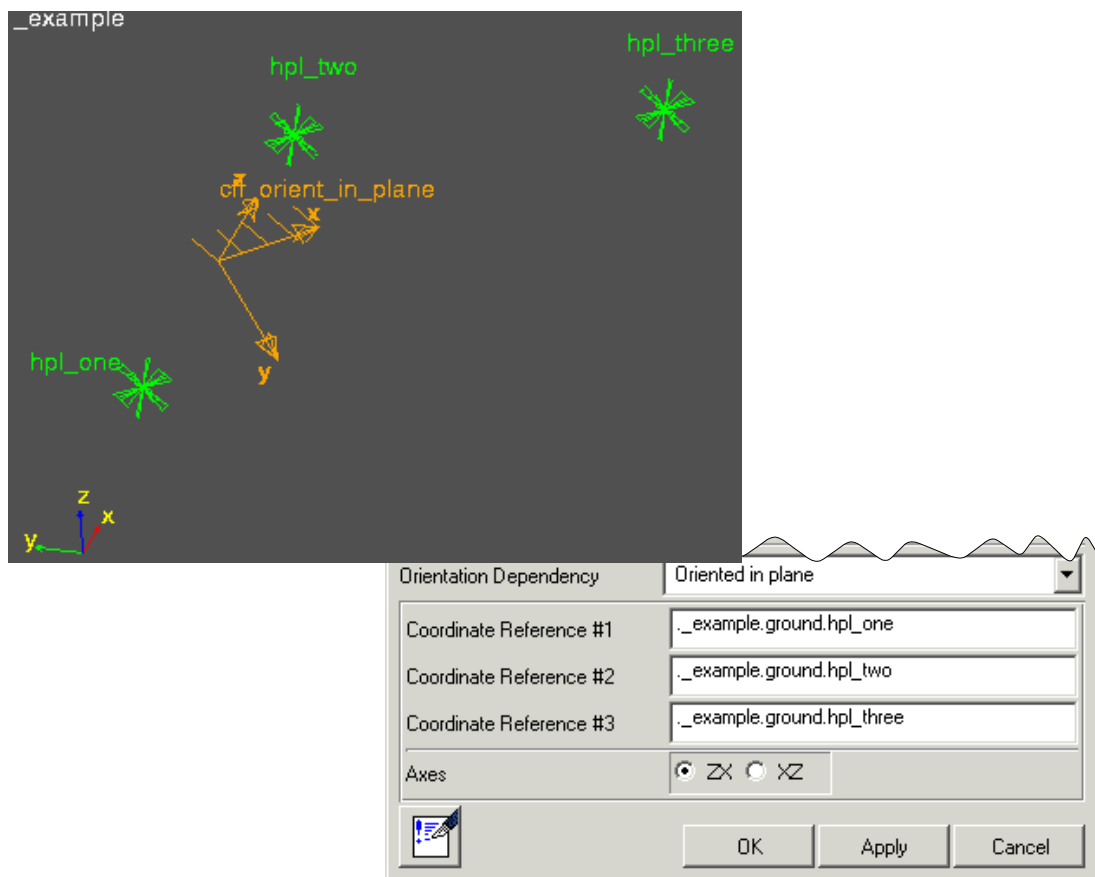
- Orients the defined axis on the entity parallel to the defined axis on the reference frame.
- In this example, the z-axis of the entity is oriented parallel to the negative x-axis on the reference frame.



# Orientation Parameterization...

## Oriented in plane

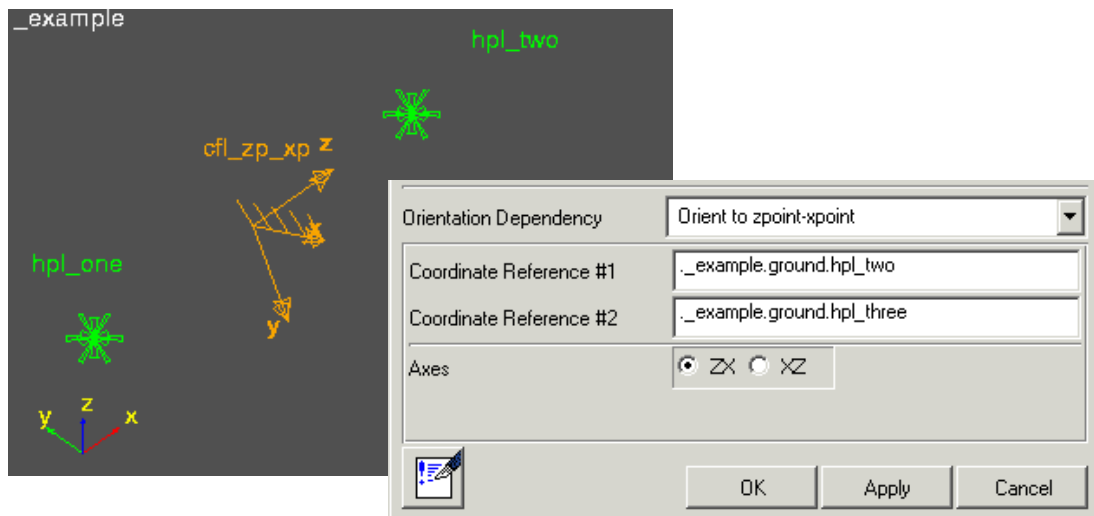
Orients the z-axis along a line defined by the first and second reference coordinates, and orients the x-axis such that the entity's zx plane lies in the plane defined by the three reference coordinates.



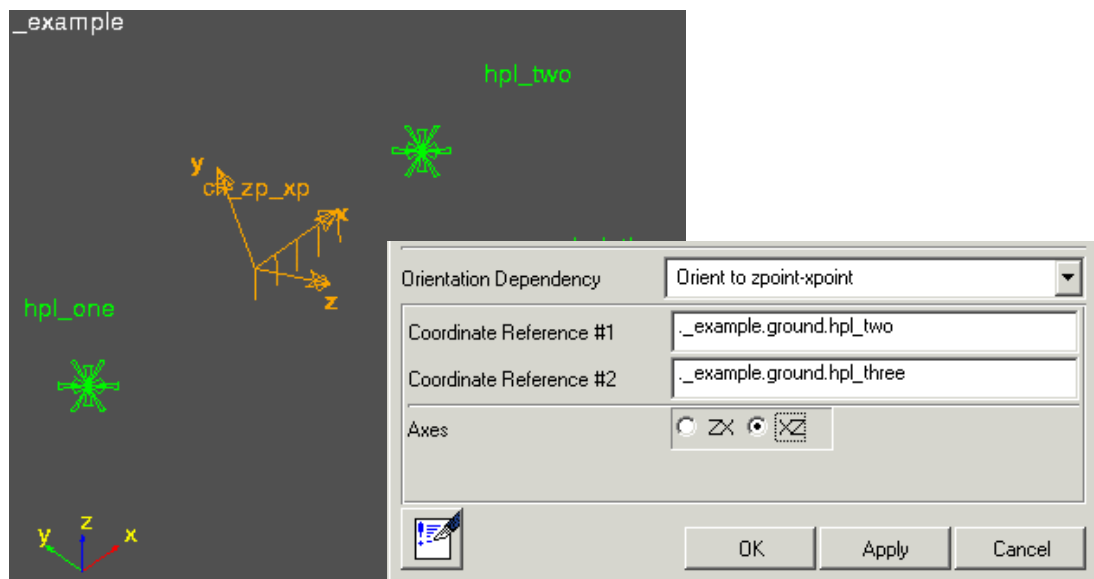
# Orientation Parameterization...

## Orient to zpoint-xpoint

Orients the z-axis towards the first reference coordinate and the x-axis towards the second reference coordinate.



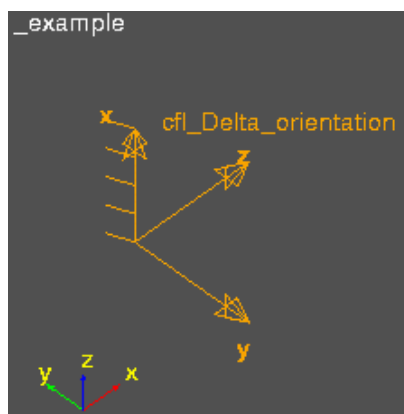
Orients the x-axis towards the first reference coordinate and the z-axis towards the second reference coordinate.



# Orientation Parameterization...

## User-entered values

Orients the construction frame using Euler angles: Z-X'-Z' incremental rotations with respect to the global construction frame (origo).



Orientation Dependency: User entered values

Orient using: ☒ Euler Angles ☐ Direction Vectors

Euler Angles: 90,90,90

X Vector: 0,0,0,1,0

Z Vector: 1,0,0,0,0

Warning icon: !

Buttons: OK, Apply, Cancel



This module provides details on creating templates, their components, and their purpose in the hierarchy of ADAMS/Car.

### What's in this module:

- Template Overview, 128
- Template Topology, 129
- File Architecture, 130
- Building a New Template, 132
- Types of Parts, 133
- Rigid Bodies (Parts), 134
- Flexible Bodies (Parts), 135
- Geometry, 136
- Attachments (Joints and Bushings), 137
- Springs, 138
- Dampers, 140
- Bumpstops and Reboundstops, 141
- Suspension Parameter Array, 142
- General Advice, 143
- Workshop 9—Template-Builder Tutorial, 144

# Template Overview

---

ADAMS/Car templates are parameterized models in which you define the topology of vehicle components. Building a template means defining parts, how they connect to each other, and how the template communicates information to other templates and the test rig. A template could represent a single set of components or a complex collection of components.

At the template level, it is not crucial that you correctly define the parts' mass properties or assign force characteristics, because this can be set at the subsystem level. It is very important, however, to correctly define part connectivity and exchange of information, because you cannot modify them at the subsystem level.

When building templates, keep in mind the assembly process. That is, make sure that your templates can communicate to each other and can communicate to the test rigs you specify. Communicators define how the different subsystems exchange information.



# Template Topology

---

**In ADAMS/Car, creating topology consists of creating elements, such as hardpoints, parts, attachments, and parameters that define subsystems, as explained next:**

- **Creating hardpoints and construction frames** - You first create hardpoints and construction frames. Hardpoints and construction frames are the ADAMS/Car elements that define all key locations and orientations in your model. They are the most elementary building blocks that you can use to parameterize higher-level entities. Hardpoint locations define most parts and attachments. Hardpoints are only defined by their coordinate location. Coordinate frames are defined by their location and orientation.
- **Creating parts** - Once you've defined hardpoints and construction frames, you will use them to create parts.
- **Creating attachments** - Finally, you create the attachments, such as joints, bushings, and parameters, which tell ADAMS/Car how the parts react in relation to one another. You can define attachments for the compliant and kinematic analysis modes. The compliant mode uses bushings, while the kinematic mode uses joints.

**Before you begin to build your template, you must decide what elements are most appropriate for your model. You must also decide which geometries seem most applicable to each part or whether you want any geometry at all. Once you've decided, you create a template and its basic topology.**

# File Architecture

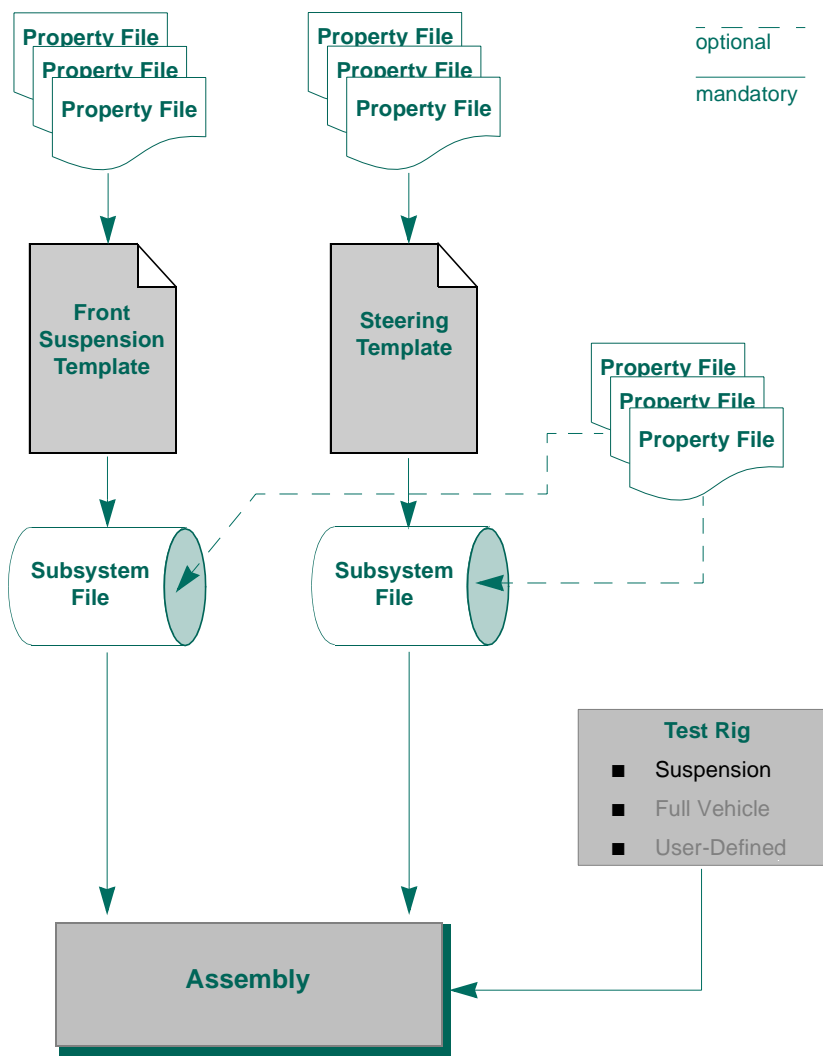
---

The file architecture of the ADAMS template-based products is comprised of the following types of files which are stored in databases:

- Property files
- Templates
- Subsystems
- Assemblies

The figure on the next page illustrates how template-based products use this architecture: a template uses property files to provide data for components such as springs, dampers, and bushings. When creating a new subsystem, you can reference the property files that the template references, or reference other property files held either in a different database or with a different file name, as indicated by the dashed lines. A collection of subsystems merged together forms an assembly.

# File Architecture...



# Building a New Template

---

To build a new template, you will create various parts, geometries, attachments, joints, and more – all of the entities necessary to create your model. To build a template, you first need to know about the available entities in ADAMS/Car, which will be described in the next sections.

Described in this module are:

- Rigid bodies (parts)
- Geometry
- Attachments (joints and bushings)
- Springs
- Dampers
- Bumpstops and reboundstops
- The suspension parameter array

Templates also require communicators, a very important part of ADAMS/Car, which allow you to pass different pieces of information from subsystem to subsystem, and necessary for assemblies. Communicators are described in the next module.

# Types of Parts

---

## Rigid bodies (parts)

- Are movable parts.
- Have mass and inertia properties.
- Cannot deform.

## Flexible bodies

- Are movable parts.
- Have mass and inertia properties.
- Can bend when forces are applied to them.

## Ground part

- Must exist in every model.
- Defines the global coordinate system (GCS) and the global origin, and, therefore, remains stationary at all times.
- Acts as the inertial reference frame for calculating velocities and acceleration.

## Mount parts and switch parts

- Mount parts are massless parts that will be replaced by other parts in the assembly process.
- Switch parts are also massless parts and act like a switch for connections. By changing the switch part, one part will connect to another.

# Rigid Bodies (Parts)

---

A rigid body is called a general part, and is abbreviated with ge[lrs] (ge for general part, [lrs] for left, right, or single). These parts can move relative to other parts and have the following properties:

- Mass
- Inertia
- Initial location and orientation [called a local body reference frame (LBRF) or a part coordinate system (PCS)]
- Initial velocities

To create a new rigid body in ADAMS/Car, go to Build->Parts -> General Part, and select either:

- **New** - Specify the location and orientation of the local body reference frame (LBRF), together with the mass properties for the body. This does not create the geometry of the part.
- **Wizard** - Select two or three hardpoints or construction frames that the part will be parameterized against. The Wizard creates the part and the geometry.

A part is used by ADAMS/Solver to perform the analysis, as opposed to a part's geometry, described in [Geometry](#) on page 136.

## Flexible Bodies (Parts)

---

You create flexible bodies by importing an external file. For information on flexible bodies, see [Using Flexible Bodies](#) on page 159.

# Geometry

---

A very important aspect of the ADAMS line of products is the concept of geometry (known as graphics in ADAMS/View). Geometry is used to enhance the visualization of a part using properties such as:

- Length
- Radius
- Width
- Thickness

As opposed to parts, geometry is not necessary to perform simulations. So, you can have a part without a geometry, but not a geometry without a part.

Note that sometimes ADAMS/Car creates the geometry for you automatically when you create a part. For example, the Wizard option does this. However, if you create a part with the New option, no geometry is created. You can then create or add geometry based on your own input.



# Attachments (Joints and Bushings)

---

Attachments, in the ADAMS/Car world, are joints and bushings, and these define how the parts in your model react to one another. You can define attachments for the compliant and kinematic analysis modes. The compliant mode uses bushings, while the kinematic mode uses joints.

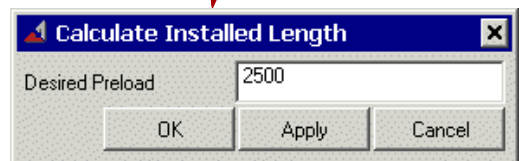
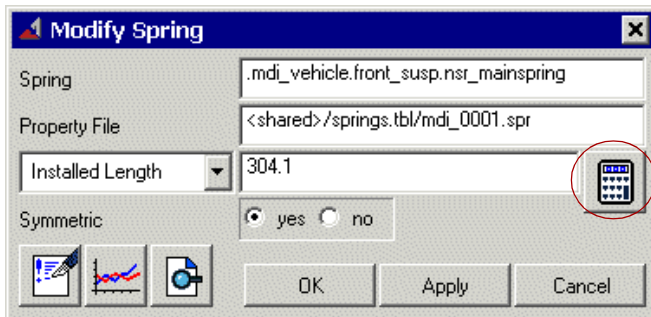
- **Bushings** - Provide three-dimensional forces and moments between parts, calculated with stiffnesses. You specify parameters, such as stiffness, preload and offset, that define bushings. Note that this is different from the BUSHING statement in ADAMS/View, which uses a constant stiffness and damping. In ADAMS/Car, the bushing effectively acts as a FIELD in ADAMS/View.
- **Joints** - Provide a kinematic constraint between parts. You specify the type of joint that is applicable to your model.

You can place both joints and bushings at the same part connections in a model. You can switch between the two attachments, activating one while deactivating the other. Thus, you can simulate kinematic and compliant analyses with the same model. You can also mix your model, using both joints and bushings for the same analysis.

# Springs

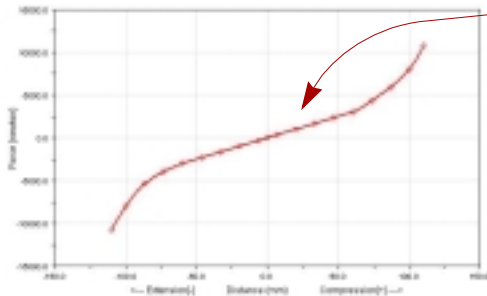
When creating a spring, you need two coordinates to attach the two ends for the spring. The coordinates can be either hardpoints or construction frames. Then, to define the spring, you must specify:

- Two bodies between which you want the force to act and two reference frames (points at which the spring is attached on the bodies).
- Installed length of the spring, which will be used to derive the design preload on the spring.
- Property file, which contains the free length information, as well as the force/deflection characteristics.



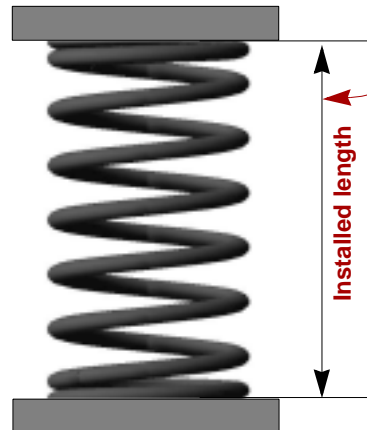
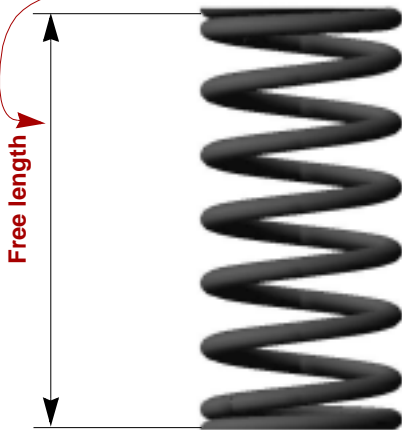
# Springs...

ADAMS/Car calculates the force exerted by the spring using the equations shown in the figure.



$$\text{Force} = -k(C - DM(i,j))$$

$$C = FL - IL + DM(i,j)^*$$



Where,

- C is a constant
- FL is the free length of the spring, as defined in the property file
- IL is the defined installed length
- $DM(i,j)^*$  is the initial displacement between the I and J coordinate reference points. If you enter a smaller value for  $DM(i,j)^*$ , ADAMS/Car calculates an increased preload for the spring; conversely, a decreased preload.
- Force represents the spring force.
- K is the nonlinear spring stiffness derived from the property file.

# Dampers

---

When creating dampers, you specify the two endpoints and the property file. Unlike a spring, a damper doesn't need a preload. ADAMS/Car also creates the geometry for the damper, like the one shown next.



# Bumpstops and Reboundstops

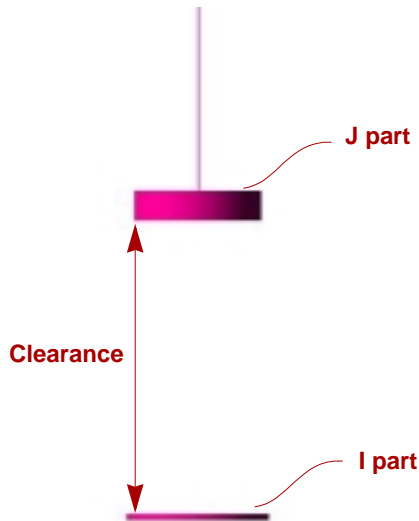
## Bumpstops

- Define a force-displacement relationship between two parts. Bumpstops act between user-specified coordinate reference points on each part, and conform to the force-displacement curve described in the designated property file. The bumpstop force is activated when the displacement between the two coordinate references exceeds a certain value, that can be defined either by either of the following:
  - ◆ **Clearance** - Defines how much part I can travel towards part J before the force activates.
  - ◆ **Impact length** - Defines how close part I can come to part J before the force activates.

## Reboundstops

- Work similarly to bumpstops, but act in rebound, instead of jounce like bumpstops.

ADAMS/Car also creates graphics for the elements shown next:



# Suspension Parameter Array

---

You access the suspension parameter array dialog box by going to Build -> Suspension Parameters -> Characteristics Array -> Set.

You first create variables defining toe and camber angles. Because these variables are commonly used for suspension analyses, ADAMS/Car creates both of them in one step. These are defined using a steer axis.

A steering axis is created using either of these methods:

- **Geometric method** - ADAMS/Car calculates the steer axis by passing a line through two noncoincident points located on the steer axis. To use the geometric method, you must identify two parts and two hardpoints that fix the steer axis. For a MacPherson strut type suspension, you might identify the wheel carrier part and a hardpoint located at the lower ball joint for the first point and the strut rod and a hardpoint located where the strut attaches to the body for the second point.
- **Instant-axis method** - ADAMS/Car calculates the left and right steer axes from the suspension's compliance matrix. While the calculation is performed numerically, it is best described in physical terms. To calculate the steer axis at a given suspension position, ADAMS/Car first locks the spring travel and applies an incremental steering torque or force. Then from the resulting translation and rotation of the wheel carrier parts, ADAMS/Car calculates the instant axis of rotation for each wheel carrier. The instant axes of rotation are the steer axes.

Finally, you set the suspension type as being independent or dependent.

## General Advice

---

Make a sketch on paper first.

Start with the hardpoints. Follow corporate naming or numbering convention where possible. This ensures that your templates can be successfully used by others within your company.

Proceed down through the Build menu.

Write comment strings when creating new objects.

Don't forget there is dialog box help using the F1 key (if installed), as well as online help.

For some components, you have more options in the modify dialog box than in the create dialog box (for example, rigid part). In that case, create the component first and then parameterize it properly using the modify dialog box.

Parts can be: rigid/flexible bodies, mounts, switch, nonlinear rods (beams in ADAMS/View terminology).

The mass properties of rigid parts can be user-entered or geometry-based.

Supported geometries are: cylinders, ellipsoids, arms, links, and outlines.

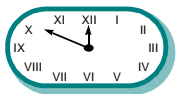
Attachments can be kinematic constraints or bushings.

Force elements include: springs, dampers, bumpstops, adjustable forces.

Advanced elements are described in the following chapters.

## Workshop 9—Template-Builder Tutorial

---



This workshop takes about two hours to complete.

Go through *Template-Builder Tutorial* in the guide, *Getting Started Using ADAMS/Car*.

As you go through the tutorial, keep in mind that when building new templates, it's good practice is to write documentation about the templates. Otherwise, it will be difficult for someone else to use your templates.



This module introduces communicators, which control how ADAMS/Car assemblies are created and how subsystems exchange information.

### What's in this module:

- Types of Communicators, 146
- Classes of Communicators, 147
- Communicator Roles, 149
- Naming Communicators, 150
- Matching Communicators During Assembly, 151
- Workshop 10—Getting Information About Communicators, 155

# Types of Communicators

---

Communicators are the key elements in ADAMS/Car that enable the different subsystems that make up your assembly to exchange information with each other and with test rigs.

A communicator is an ADAMS/View variable. A communicator contains either a(n):

- Object (for example, a part, variable, marker, or joint)
- Real value (for example, x,y,z location)
- String

## Types of Communicators

An assembly requires two directions of data transfer between its subsystems. To provide for these two directions of data transfer, ADAMS/Car has two types of communicators:

- **Input communicators** - Demand information from other subsystems or test rigs.
- **Output communicators** - Provide information to other subsystems or test rigs.

Think of an input communicator as an electrical plug, and an output communicator as a power strip. The electrical plug requires electricity from the power strip.

For example, a mount communicator in the rack and pinion steering templates outputs the rack part name so that tie rods of suspension templates can attach to the rack. In addition, a mount communicator in the steering template inputs a part name to determine where to attach the steering column to the body.

# Classes of Communicators

The class of a communicator indicates the kind of information it exchanges. For example, communicators of the class **hardpoint** exchange a location through a **hardpoint** name and a **part** name. The classes of communicators and the information that each class exchanges are listed in the table below. The classes apply to both input and output communicators.

The class:	Exchanges:
Mount	Part name to provide connections between subassemblies. As a shortcut, the template-based products also automatically create input mount communicators when you create a mount part.
Marker	Hardpoint and part name to provide both location and part information. If the hardpoint is part of a symmetrical pair, the template-based products create an input communicator for each hardpoint in the pair.
Joint	Joint name.
Joint-for-motion	Joint name.
Bushing	Bushing name.
Array	ADAMS/Solver array name.
Spline	Spline name.
Differential	Differential equation name.
Solver variable	ADAMS/Solver variable name. You must use an ADAMS/Solver variable and not an ADAMS/View variable. Unlike an ADAMS/View variable, an ADAMS/Solver variable's computation occurs during analysis. ADAMS/Car generates ADAMS/Solver variables as state variables.
Location	The location of the named hardpoint or construction frame. If the hardpoint is part of a symmetrical pair, the template-based products create two input communicators, one for each hardpoint in the pair.
Motion	Motion name.
Part	Part name.
Orientation	The orientation of the named construction frame.
Real parameter	A parameter variable name of the type real.
Integer parameter	A parameter variable name of the type integer.

## Classes of Communicators...

---

A communicator can be either single or be part of a symmetrical pair, either left or right. Entity classes (array, differential equation, motion, parameter variable, solver variable, and spline) have no symmetry and, therefore, are always single, by default.

# Communicator Roles

---

Each communicator has a minor role. A minor role defines the communicator's position in the assembly. ADAMS/Car provides you with five default minor roles:

- Front
- Rear
- Trailer
- Inherit
- Any

If you select *inherit*, the minor role of the communicator will become that of the subsystem using the template.

You can define a communicator's minor role when you create it. For example, if you want to provide input to or output from subsystems of specific roles, then you set the minor role for communicators when you create them. We recommend, however, that you do not set a communicator's minor role. Instead, let the subsystem do it. For example, a suspension template might be used to define either a *front* or *rear* suspension subsystem. By letting the subsystem determine the minor role, the assembly process attaches a steering system to the front suspension and not to the rear.

# Naming Communicators

---

After you create a communicator, ADAMS/Car assigns a prefix to the name. For example, it creates a prefix, *ci\_* where: *ci* indicates it is an input communicator. If it were an output communicator, ADAMS/Car would use *co*. */* indicates it is for the left side of a symmetrical pair. If it were for the right side, ADAMS/Car would use an *r* (*cir*). If it were a single communicator, it would have an *s* (*cis*).

If you create a mount part, ADAMS/Car automatically creates an input communicator of the class *mount*. It uses the name of the mount part as the name of the communicator and appends the prefix *ci[lrs]\_* to it, depending on whether or not it is a *left*, *right*, or *single* communicator. For example, if you create a mount part of *mtl\_rack\_mount*, ADAMS/Car creates an input communicator with the name *ci\_l\_rack\_mount*, where the */* indicates it is for the left side. Note: You cannot create a mount input communicator by itself. You must create a mount part, and ADAMS/Car will automatically create the communicator for you.

As you name communicators, you should ensure that any input and output communicators that exchange information have matching names. For example, the name you give to communicators that exchange a part name during assembly might be *ci[lrs]\_strut\_mount* and *co[lrs]\_strut\_mount*. In addition, if you are working with MDI templates, you must ensure that you use the same naming conventions as the MDI templates.

# Matching Communicators During Assembly

---

For a pair of communicators to exchange information during assembly, the communicators must:

- 1 Have matching names.
- 2 Be of opposite types (one input, one output).
- 3 Be of the same symmetry type (*left*, *right*, or *single*).
- 4 Be of the same class (exchange the same type of information); for example, *mount*.
- 5 Have the same minor role or be assigned a role of *any*.

If an input communicator does not have a corresponding output communicator, ADAMS/Car returns a warning message, and, if the input communicator belongs to the class *mount*, ADAMS/Car assigns the mount part to ground. ADAMS/Car gives you a warning message, because your input communicator does not have the information it requires and, thus, your assembly may not have all of the information it needs to work properly.

On the other hand, if an output communicator is not linked up with one or more input communicators, you will not get a warning upon assembling your subsystems, because simply publishing information has no direct effect on the operation of your assembly.

You can still analyze the model if it does not have matching communicators. In fact, you may find this helpful if you want to run an analysis of a subsystem without attaching another subsystem to it.

# Matching Communicators During Assembly...

For example, the following pairs of input and output communicators match and exchange a part name during assembly.

The pair:	Belongs to the class:	From minor role:	To minor role:
cil_strut_mount	mount	front	
col_strut_mount	mount		front
cil_strut_mount	mount	any	
col_strut_mount	mount		front
cil_strut_mount	mount	front	
col_strut_mount	mount		any

In addition, an input communicator can only be matched with one output communicator, but one output communicator can be matched with an unlimited number of input communicators. This is because input communicators need all of the information provided by a single output communicator, and if there is more than one specified, the input communicator will not know which one to choose.

Alternatively, output communicators just publish information, and can give this information to whatever input communicator needs it. You should always check the warning messages during the assembly, especially if the warnings refer to an input communicator of class *mount* that does not get assigned and is, therefore, attached to ground.



# Matching Communicators with Test Rigs

When you create a template, you must meet the following conditions to ensure that analyses will work with your new template:

- The template must be compatible with other templates and with the test rigs, for example, the `.__MDI_SUSPENSION_TESTRIG`.
- The template must contain the proper output communicators.

If the template is a suspension template (that is, its major role is *suspension*), it must contain a suspension parameters array. The suspension parameters array identifies to the suspension analysis how the steer (kingpin) axes should be calculated and whether the suspension is independent or dependent.

For example, for a suspension template to be compatible with `.__MDI_SUSPENSION_TESTRIG`, the suspension template must contain the following output communicators. In the table, the prefix `[lr]` indicates that there is both a left and right communicator of the specified name.

## Output Communicators in Suspension Template

The communicator:	Belongs to the class:	From minor role:
co[lr]_suspension_mount	mount	inherit
co[lr]_wheel_center	location	inherit
co[lr]_toe_angle	parameter_real	inherit
co[lr]_camber_angle	parameter_real	inherit

## Matching Communicators with Test Rigs...

---

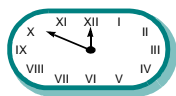
The *co[lr]\_suspension\_mount* output communicators publish the parts to which the test rig wheels should mount. As you create these communicators, make sure that you set their minor role to *inherit*. By setting the minor role to *inherit*, the communicator takes its minor role from the minor role of the subsystems that use your suspension template.

The *co[lr]\_wheel\_center* output communicators publish the location of the wheel centers to the test rig so the test rig can locate itself relative to the suspension. As you create these types of communicators, make sure that you also leave their minor role set to *inherit*.

The toe and camber communicators (*co[lr]\_toe\_angle* and *co[lr]\_camber\_angle*) publish to the test rig the toe and camber angles set in the suspension, so the test rig can orient the wheels correctly.

For more information, see the guide, *Building Templates in ADAMS/Car*.

# Workshop 10—Getting Information About Communicators



This workshop takes about one half hour to complete.

In this workshop, you learn how to perform tests and display information that will help you understand communicators.

## Getting communicator information

### To get information about a communicator in your template:

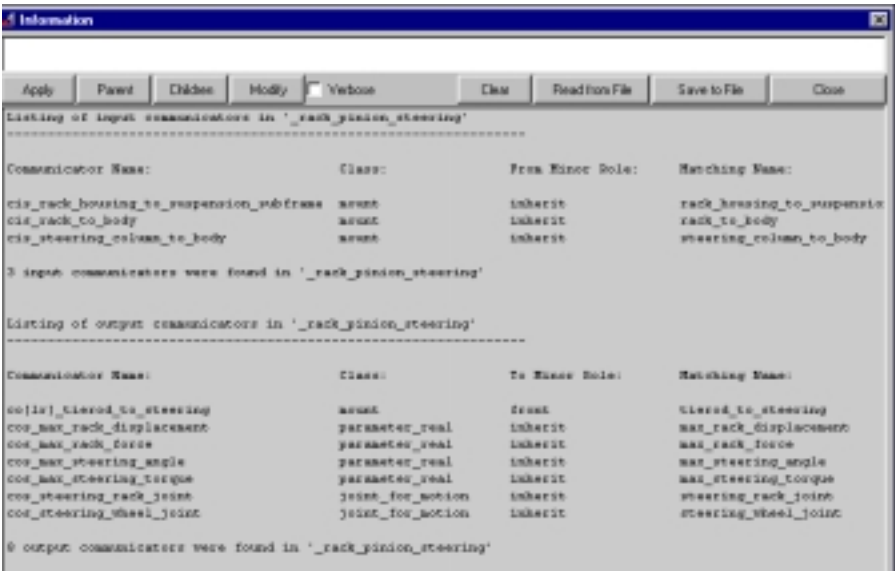
- From the **Build** menu, point to **Communicator**, and then select **Info**.

**Note:** In Model Names, your current open template will automatically be selected.

### To list information about all the communicators in your template:

- 1 In the **Communicators Info** dialog box, set **Entity** to **All**.
- 2 Select **OK**.

The Information window lists information for all communicators:



# Workshop 10—Getting Information About Communicators...

## Checking communicators

In this section, you set up your workspace such that you can see how the communicators in your rack and pinion template match up with those in a suspension template.

You will open an existing template you know your new template will be connected to when you run analyses. This allows you to see which communicators the other template requires to function together with your template. You need to open a suspension template to do this.

**Note:** Make sure that all templates you want to test are open in your ADAMS/Car session.

### To open a template:

- 1 From the **File** menu, select **Open**.
- 2 Right-click the **Template Name** text box, point to **Search**, and then select `<shared>\templates.tbl`.
- 3 Double-click `_double_wishbone_torsion.tpl`.

The template opens as a new model, unassociated with your rack and pinion template, which is still open.

The double-wishbone suspension appears in your workspace.

### To perform a communicator test:

- 1 From the **Build** menu, point to **Communicator**, and then select **Test**.
- 2 Right-click the **Model Names** text box, point to **Model**, point to **Guesses**, and then select `_double_wishbone_torsion`.
- 3 Repeat [Step 2](#) to select `_rack_pinion_steering`.

## Workshop 10—Getting Information About Communicators...

- 4 In the **Minor Roles** text box, enter the minor roles of the communicators. You must enter one minor role for each model or test rig that you select.

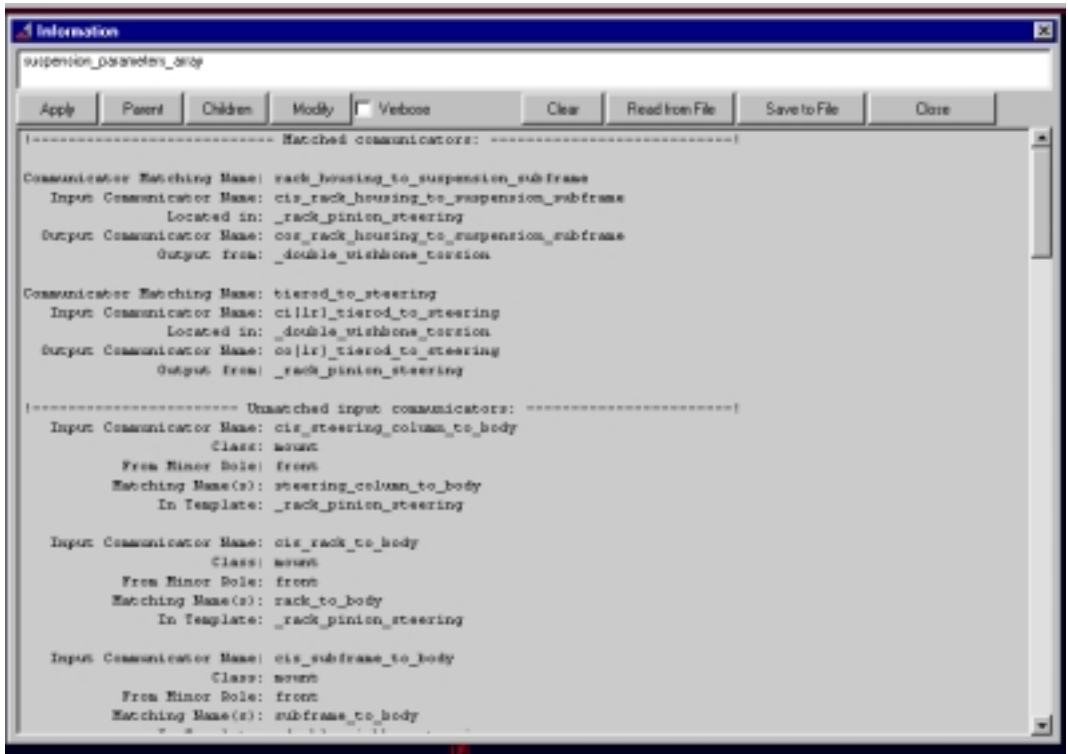
**Note:** Each communicator has a minor role, which by default is one of the following: any, front, rear, trailer, or inherit. The inherit minor role specifies that when ADAMS/Car creates a subsystem from the template, the communicator should inherit the subsystem's minor role. Since when you test a template's communicators, the inherit minor role is still undefined, entering minor roles in the Minor Role text area provides the communicators with their minor role. For example, if you assign the template `susp_02`, a minor role of front in the Minor Roles text area, the communicator test also changes the minor role of any communicators in `susp_02` whose minor role is inherit to the role of front.

- 5 If you've used the Information window before, select **Clear Information Window**.
- 6 If you want to save the results, select **Save Test Results to File**.
- 7 In the **File Name** text box, enter a file name.
- 8 Select **OK**.

The Information window, as shown next, lists the communicators that match other communicators and those that do not. It shows the matched communicators followed by the unmatched communicators. The lists include the names of the input and output communicators and the names of the templates to which they belong. Often, you'll see many communicators that are unmatched. Many of these communicators are related to subsystems or test rigs that you do not currently have open.

In this case, note that the communicators `rack_housing_to_suspension_subframe` and `tierod_to_steering` are matched. However, communicators such as `rack_to_body`, and `max_rack_force` are unmatched, because the particular templates which require this information were not selected (and unopened).

# Workshop 10—Getting Information About Communicators...



To get to know the topology of the communicators within a template, study the inputs and outputs and look to see how they link with other templates. When you create communicators in a new template, a good way to create the communicators is to open an existing template and create the same communicators.

A final tip is to remember that output communicators publish information. Therefore, they are passive and do not affect subsystem behavior. However, input communicators search for information. They affect your simulation, particularly if they do not match.

For more tips on how to get information about communicators, see Article 8924, *Investigating Existing Templates* in the Knowledge base, at: <http://support.adams.com/kb/faq.asp?ID=kb8924.html>.

In this module, you will learn how to create flexible bodies in your models, as well as how to swap a rigid body for a flexible body.

**What's in this module:**

- Flexible Body Overview, 160
- Limitations of Flexible Bodies, 161
- Getting Flexible Bodies, 162
- Workshop 11—Flex Tutorial, 163

# Flexible Body Overview

---

ADAMS/Flex uses an assumed-modes method of modeling flexible bodies, called modal flexibility. Modal flexibility assigns a set of mode shapes to a flexible body. This modal method of modeling flexibility can be very useful in problems that are characterized by high elasticity and moderate deflections. That is, deflections less than 10% of a characteristic length of the body.

By integrating flexible bodies into your model, you can:

- Capture inertial and compliance properties during handling and comfort simulations.
- Predict loads with greater accuracy by allowing ADAMS to account for flexibility during simulations.
- Study deformation.
- Examine the linear system modes of a flexible model when you use ADAMS/Flex with ADAMS/Linear.

You should use flexible bodies wherever you expect component flexibility to affect the dynamic behavior of your model or when you require accurate information about the deformations of a component in your model.



# Limitations of Flexible Bodies

---

When you use flexible bodies, remember that flexible body deformations are a linear combination of deformation shapes. Consequently, take special precautions when modeling higher order deformations, such as those that occur when deformations are large, or when attempting to correctly model centrifugal stiffening of rotating systems. You can overcome these limitations by dividing a flexible body into multiple flexible bodies and assembling them in ADAMS/Car.

Also, note that flexible bodies are not parametric. If you want to substitute a new flexible body in your system, you must create a new flexible body.

# Getting Flexible Bodies

---

## Two ways to create flexible bodies

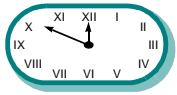
- Importing modal neutral files (.mnf) - To create a new flexible body, go to Build -> Part -> Flexible Body. ADAMS/Car imports the .mnf file and creates the flexible body.
- Creating .mnf files with Autoflex - With the additional module (it requires a separate license), a flexible body can be generated without access to an external FEA package. Specify the cross section, center line, and attachment points. This tool generates the flexible body, just like importing an .mnf file.

## Flexible bodies require a special part

- Creating interface parts - At every location on the flexible body where you intend to attach a joint or a force, you should place an interface part. Place the interface parts on nodes of the flexible body. This works as a dummy part that is attached to the flexible body with a fixed joint. You can apply other elements to these interface parts.

**Note:** When picking nodes for the interface parts, the interface part dialog box needs left and right nodes and then creates left and right interface parts.

# Workshop 11—Flex Tutorial



This workshop takes about one and a half hours to complete.

In this workshop, you swap a rigid lower control arm with a flexible control arm.

First, in ADAMS/Car Template Builder, open the double-wishbone suspension template.

## Creating a flexible body

Create the flexible body on top of the rigid part, and swap the connections from the rigid to the flexible body. When nothing is referenced by the rigid part, you can delete it.

### To create the flexible lower control arm:

- 1 From the **Build** menu, point to **Parts**, point to **Flexible Body**, and then select **New**.
- 2 Enter the following user-entered location for the left control arm: **-266.0, 0.0, -30.0 mm**.  
The symmetry rule automatically puts the right control arm at the same location, but with the opposite y value. The flexible body uses the local part reference frame (LPRF) that was set in the FEA program to be the origin.
- 3 Point to the files **LCA\_left\_shl.mnf**, and **LCA\_right\_shl.mnf**, located in the shared database.
- 4 Select the color you want on the graphics for the flexible body. The graphics for the flexible body reside in the .mnf file. To get a nice flexible body, you may need a very large .mnf file.
- 5 Select **OK**.

ADAMS/Car displays the flexible bodies on top of the rigid bodies when in wireframe mode.

## Creating interface parts

Now you create interface parts at every location where you have a connection to another element in the model. In this case, you have two connection points to the chassis. If you just want one revolute joint to the chassis, you only need one interface part for the revolute joint, but if you want a bushing for the front and rear attachment points, you need two interface parts, one at each bushing. You also need one attachment to the damper, and finally, you need one attachment for the upright. You then substitute the corresponding interface parts that you create for the rigid parts in the attachments.

### To create interface parts:

- 1 From the **Build** menu, point to **Parts**, point to **Flexible Body**, point to **Interface Part**, and then select **New**.

Supply the following data:

- The name of the interface part.
- If it's a left, right, or single.
- What flexible body it's attaching to.
- Left and right node ID, or the single node ID.

Use the Pick command to do this; use the mouse to highlight nodes on your flexible body.

- Geometry radius. An interface part is represented by a sphere, so this radius is the radius of the sphere.
- Color of the sphere.

- 2 Repeat this for each of the attachment points.

# Workshop 11—Flex Tutorial...

---

## Altering connections

You now change the connections from the rigid body to the interface part by modifying each attachment.

### To move the connections:

- 1 Right-click on the front bushing that connects the lower control arm to the subframe and select **Modify**.
- 2 In the **I Part** text box, replace the part `gel_lower_control_arm` with the name for your left interface part.
- 3 Select **OK**.
- 4 Make sure you modify all attachments at that location. For example, if you have a joint that is only active in kinematic mode, you also need to modify the bushing at the same location. The graphical topology might help you find all of the connections.
- 5 To get to the graphical topology, go to **Tools**, point to **Database Navigator**, and then select **Graphical Topology**.
- 6 When you're done, delete the rigid lower control arm. If you forgot any attachments, ADAMS/Car will issue a warning and prompt you to select one of the following:
  - **Continue** - ADAMS/Car deletes the entity you selected anyway, and any other objects dependent on your selected object.
  - **Highlight & List Dependents** - ADAMS/Car lists all dependencies that still exist in the model. These dependencies will also be highlighted.
  - **Cancel** - ADAMS/Car takes no action. If you select **Cancel**, you must delete the dependencies manually before ADAMS/Car can delete this entity.

## Workshop 11—Flex Tutorial...

---

**7** Edit the switch part.

This model contains a switch part that is used for an anti-roll bar (ARB). Therefore, you must replace the `gel_lower_control_arm` part with the appropriate interface part in the switch part. Because the other part in the switch part list is the upright, you'll use the interface part used to connect the upright, as well as the same location.

**8** The suspension parameter array also contains the part: `gel_lower_control_arm` which will be replaced by the appropriate interface part. However, the suspension parameter array can only be set for one configuration, so you have to delete it first and then set it with the new interface part.

Make the following changes to the suspension parameters array (geometric):

- I Part:                                      interface part
- J Part:                                      gel\_upper\_control\_arm
- I Coordinate Reference:    hpl\_lca\_outer
- J Coordinate Reference:    hpl\_uca\_outer

**9** Delete the part `gel_lower_control_arm`.

**10** Switch to Standard Interface and create a new subsystem and suspension assembly.

**11** Run a suspension analysis.

**12** If time allows, perform the [Tutorial for ADAMS/Flexible Body Generator](#), on page 93 in the guide, *Getting Started Using ADAMS/Car*.

This module introduces requests, which are the primary method of output in ADAMS/Car.

**What's in this module:**

- [Creating New Requests, 168](#)
- [Types of Requests, 169](#)

# Creating New Requests

To create new requests, in Template Builder, go to Build -> Requests -> New. You can only create requests in Template Builder.

**Modify Request**

Request Name:

Comment:

Define Using Function Expression:

F1:

F2:

F3:

F4:

F5:

F6:

F7:

F8:

Title:

Component Attributes

Result Set Name:

MAG	<input type="text"/>	<input type="text" value="no units"/>
X	<input type="text" value="longitudinal"/>	<input type="text" value="velocity"/>
Y	<input type="text" value="lateral"/>	<input type="text" value="velocity"/>
Z	<input type="text" value="vertical"/>	<input type="text" value="velocity"/>
AMAG	<input type="text"/>	<input type="text" value="no units"/>
R1	<input type="text" value="roll"/>	<input type="text" value="angular velocity"/>
R2	<input type="text" value="pitch"/>	<input type="text" value="angular velocity"/>
R3	<input type="text" value="yaw"/>	<input type="text" value="angular velocity"/>

The results set names appears in the Request list in ADAMS/PostProcessor

The other attributes appear in the Component list

Request		Component	
chassis_accelerations	(Chassis	longitudinal	
chassis_displacements	(Chassis	lateral	
chassis_velocities	(Chassis	vertical	
dal_ride_damper_data		roll	
dar_ride_damper_data		pitch	
differential		yaw	
driver_demands	(Driver		



# Types of Requests

---

## Types of requests:

- User subroutine
- Type and markers based
- Function expression

Note that the name file (.nam) acts as key by matching ID's with request names. You can edit the .nam file to change the name of certain requests and the way they appear in ADAMS/PostProcessor.

For details, see the REQUEST statement in the guide, *Using ADAMS/Solver*.



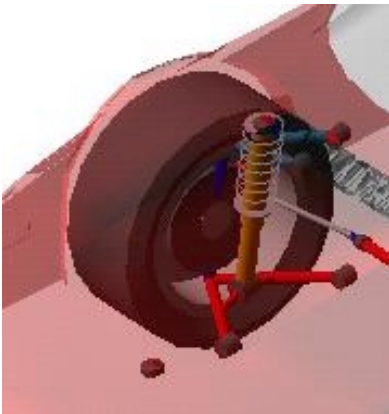
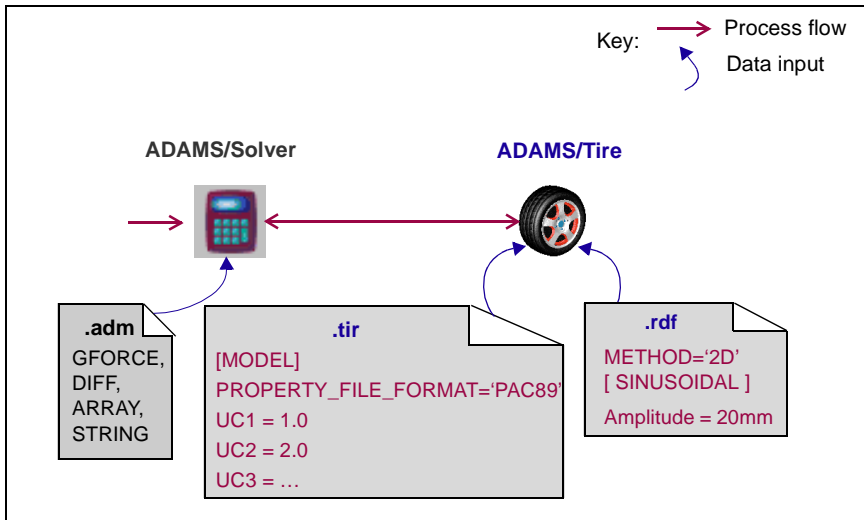
This module provides an overview of the calculation of tire forces and the available models.

### What's in this module:

- Tire Overview, 172
- ADAMS/Tire Modules, 173
- Tire Models, 175
- Tire Analyses, 176
- Workshop 12—Building a Wheel Template, 177

# Tire Overview

ADAMS/Tire calculates the forces and moments that tires exert on the vehicle as a result of the interaction between the tires and road surface.



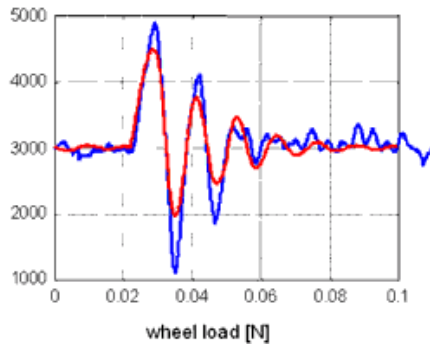
To perform tire calculations, you need one or more of the following modules:

- **ADAMS/Tire Handling module** - Incorporates the following tire models for use in vehicle dynamic studies:
  - ◆ Delft-Tyre model from the Netherlands Organization for Applied Scientific Research (TNO) (DTM 95, DTM 96, our most advanced models)
  - ◆ Pacejka '89 and Pacejka '94 models
  - ◆ Fiala tire model (least advanced)

ADAMS/Tire Handling uses a point-follower method to calculate tire normal force and is limited to two-dimensional roads.

- **ADAMS/Tire Durability module** - Uses a three-dimensional equivalent volume method to calculate tire normal force on three-dimensional roads for use in predicting vehicle loads for durability studies. When you purchase ADAMS/Tire Durability separately, you can use only the Fiala model to calculate tire handling forces and moments.
- **ADAMS/Tire FTire module** - The FTire module is the latest addition to ADAMS/ Tire. A new tire model for durability and ride and handling applications, FTire:
  - ◆ Offers an effective compromise between model fidelity and detail, and computational speed.
  - ◆ Provides valid results up to 120 Hz in the frequency domain.
  - ◆ Lets you easily derive model parameters from tire measurement data.
  - ◆ Provides valid results for short obstacles with wavelengths down to half the size of the tire-road contact patch.

- ◆ Provides highly accurate solutions when passing through potholes and over cleats.



- FTire is a 2½D nonlinear tire model. The tire belt is represented as a ring of small elements. Typically there are 50 to 100 elements. The elements are connected to each other through stiff springs and dampers. This ring of elements can be bent in any direction relative to the wheel rim.
- FTire conforms to the TYDEX Standard Tire Interface (STI).

When you purchase ADAMS/Tire Durability with ADAMS/Tire Handling, you can use the Delft-Tyre, Pacejka '89, Pacejka '94, or Fiala models to calculate the tire handling forces and moments (lateral force, longitudinal force, aligning torque, and so on).

## Tire data

Tire data is essential to obtain accurate tire forces during a simulation. If you use the Fiala model, you can generate the tire property file by hand. If you use a different tire model, you must use a fitting routine to obtain the coefficients for the tire property file. This is usually done by the testing facility that tests the physical tire. Unless the tire you want to use is tested already, a test must be performed to obtain the tire data necessary for a tire property file.

Table 2. ADAMS/Tire Modules and Features

ADAMS/Tire Features:	ADAMS/Tire Modules:		
	Handling	Durability	Handling and Durability
Handling Force Models			
Fiala	■	■	■
Pacejka ‘89	■		■
Pacejka ‘94	■		■
Delft MF-Tyre (Pacejka ‘96)	■		■
Normal Force Models			
Linear spring force	■	■	■
Road Contact Models			
2D point follower (flat surface)	■	■	■
3D equivalent volume (discrete surface)		■	■
Miscellaneous			
Uses ADAMS/Solver GFORCE interface	■	■	■
Meets STI v. 1.4 architecture standards	■	■	■
Supports user-written STI tire models	■	■	■

# Tire Analyses

---

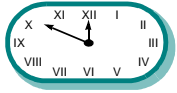
ADAMS/Tire reads the model block [MODEL] portion of the tire (.tir) and road data file (.rdf) to determine which tire model and road contact model to use when calculating the tire forces and moments.

During an analysis, ADAMS/Tire calls the tire model to calculate the tire forces and moments. In this process, the tire model calls the road model to calculate the tire contact point and local road normal, and uses these values with the tire velocity and orientation to calculate the forces and moments.

ADAMS/Tire then returns the forces and moments to ADAMS/Solver, which applies the forces and moments to the wheel part.



# Workshop 12—Building a Wheel Template



This workshop takes about one hour to complete.

In this workshop, you create a wheel template that sets up the interface to your tire model.

## Creating a template

### To create the wheel template:

- 1 From the **File** menu, select **New**.
- 2 Name the template anything you want.
- 3 Set **Major Role** to **wheel**.
- 4 Select **OK**.

## Creating communicators

### To create communicators:

- 1 Create two input communicators for toe and camber angles. They should be of type **parameter\_real** and be named **toe\_angle** and **camber\_angle**.
- 2 Create an input location communicator that receives the wheel center location. This communicator positions the wheel at the location dictated by the suspension. The corresponding output communicator, **wheel\_center**, resides in the MacPherson suspension you created in [Workshop 9—Template-Builder Tutorial](#), on page 144.

# Workshop 12—Building a Wheel Template...

---

## Creating construction frames and mount parts

### To create the construction frame and mount part:

- 1 Create a construction frame to be used as the spin axis. Locate this construction frame on an input communicator, which should match the output communicator from the suspension. This construction frame should also be dependent on two communicators for toe and camber angles. (See the different orientation dependency options.) Verify that the spin axis is defined right, such that the z-axis points out from the vehicle (this should be done automatically for you).
- 2 Create the mount part that will attach to the suspension.

## Creating the wheel part

### To create the wheel part:

- 1 From the **Build** menu, point to **Wheel**, and then select **New**.
- 2 Create the wheel part with the following data:
  - Mass: 20.0 kg
  - lxx lyy: 5E4 kg-mm\*\*2
  - lzz: 1E4 kg-mm\*\*2
  - Property File: mdi\_tire01.tir
  - Contact Type: handling
  - Coord ref loc: cfl\_spin\_axis
  - Location: 0, 0, 0 mm
  - Coord ref ori: cfl\_spin\_axis
  - Orientation: 0, 0, 0 deg


**Note:** ADAMS/Car automatically creates a pair and sets the tire geometry in the property file.

# Workshop 12—Building a Wheel Template...

---

## Viewing tire geometry

### To view the tire geometry:

- 1 To view the tire property file, select the **View File** tool .
- 2 Search for the data block named **DIMENSIONS**.

## Connecting mount and wheel parts

### To connect the mount part to the wheel part:

- Create a fixed joint between the tire and the mount part located at the `cfl_spin_axis`.

## Testing communicators

### To test communicators:

- Test the communicators and select this tire template and the suspension template you intend to use (make sure that you have these templates open in your session).

Note that this template has been developed to be used with the MacPherson suspension template you created in [Workshop 9—Template-Builder Tutorial](#), on page 144. If you use this template with another suspension, there might be other communicators you must match.

## Workshop 12—Building a Wheel Template...

---

Before you build or modify templates, it is important that you know how templates are built and organized. This includes not only how parts are connected with joints, motions, and forces, but also how the templates exchange information with communicators. This chapter introduces methods of exploring templates to enable you to modify existing templates for your own use and create new templates which are compatible with others.

### What's in this module:

- [Investigating Templates, 182](#)
- [Understanding Templates, 183](#)
- [About the Database Navigator, 184](#)
- [Workshop 13—Exploring and Completing Templates, 186](#)

## Investigating templates primarily involves:

- Getting information about the template components. Once you understand what components make up a template and how they relate to each other, you can modify the components to make your template unique.
- Getting information about communicators. Understanding how the template works internally is not enough: you must also understand how it communicates with other templates and test rigs. This understanding will help you connect your templates correctly.

# Understanding Templates

---

## Guidelines to help you understand how a template is built:

- 1 Right-click the component for which you want to obtain information, point to the component name, and then select **Modify**.

The Modify Component dialog box appears. Note the information displayed in the dialog box, such as attachment and positioning information.

- 2 You can use the Template Builder's Build menu to list all components by type, as well as their specifications, such as mass, location, orientation, and so on. When requesting information for most components, your template-based product displays the Entity Information dialog box. From this dialog box you can select the type of component on which you request information.
- 3 When requesting information for communicators, your template-based product displays the Communicators Info dialog box. Use the Communicator Info dialog box to list the communicators in different templates and test rigs.
- 4 Use the Database Navigator to explore your template.

## Displaying the Database Navigator

You can display the Database Navigator by doing any of the following:

- From the Tools menu, select Database Navigator.
- From the Edit menu, execute an editing command, such as Modify, when no object is currently selected.
- From the Edit pop-up menu, request to view information about an object using the Info command.
- Using the Browse command, browse for the name of an object to enter in a dialog box.

The Database Navigator has several modes in which you can display object information. You can set it to just let you browse for objects or you can set it to view information about the objects, such as view how an object relates to other objects, and view dependencies. The Database Navigator only displays the types of objects that are appropriate for the command you are executing. For example, if you are renaming a model, it only displays models in your database. On the other hand, if you are searching for any modeling object in the database, it displays all types of modeling objects. You can also set a filter for the types of objects that the Database Navigator displays.

The Database Navigator shows objects in their database hierarchy.



# About the Database Navigator...

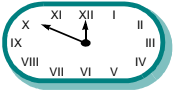
---

## Viewing Model Topology

You can use the Database Navigator to display information about the parts in your model. You can:

- View the topology of a model - When you request information about a model's topology, ADAMS/View determines what constraints are owned by the model and what parts the constraints connect. There are two different ways in which you can show the part connection information: by part and by connection. For more information on the two ways, see the guide, *Learning ADAMS/View Basics*.
- Graphically view the topology - In graphical topology, the Database Navigator displays a representation of the selected part and shows its connections to other parts. The connections represent the joints or forces between the parts. Each time you select a different part in the tree list of the Database Navigator, the graphical display changes to show the selected part at the center of the display.

# Workshop 13—Exploring and Completing Templates



This workshop takes about one hour to complete.

This workshop tests some of the topics about template building, including defining communicators and joints, and investigating your template with the database navigator.

We recommend that whenever possible, you modify existing templates, rather than create new ones. To be able to modify existing templates and to customize them for your use, you must be able to understand them very well.

## Defining your template

### To define your template:

- 1 Copy `_steer_training.tpl` to the `template.tbl` directory of your choice (for example, place it in `private.cdb/templates.tbl`).
- 2 Open `_steer_training.tpl` in template-builder mode.

A rack and pinion steering template appears. The template is incomplete: it needs joints to be defined, as well as communicators and mount parts. Use the Database Navigator to investigate the template by looking at the parts and icons, and try to determine what yet needs to be defined for this template.
- 3 Make the necessary changes to define your template properly. To initiate exploration of your model and challenge yourself to determine what entities still need to be defined, first see [General steps to define your template](#) on page 187. If you have trouble or would like to check your work, see [Detailed steps to define your template](#) on page 187 to determine what changes should be made.

# Workshop 13—Exploring and Completing Templates...

---

## General steps to define your template

### To define your template:

- 1 Constrain the motion of the steering columns to each other.
- 2 Constrain the motion of the steering wheel to the steering column (Hint: Use a gear to do this. To learn more about gears, press **F1** when the cursor is in the window, and then click anywhere on the dialog box.)
- 3 Constrain the motion of the steering shaft to the rack housing.
- 4 Constrain the motion of the steering shaft to the rack (Hint: Use a gear to do this).
- 5 Constrain the motion of the rack to the rack housing.
- 6 Create a mount part for the rack housing.
- 7 Make sure the rack will be able to connect to the MacPherson template you created earlier. If necessary, create any mount parts or communicators.
- 8 Check that the steering column housing will mount properly. If necessary, create any mount parts or communicators.

## Detailed steps to define your template

### To define your template:

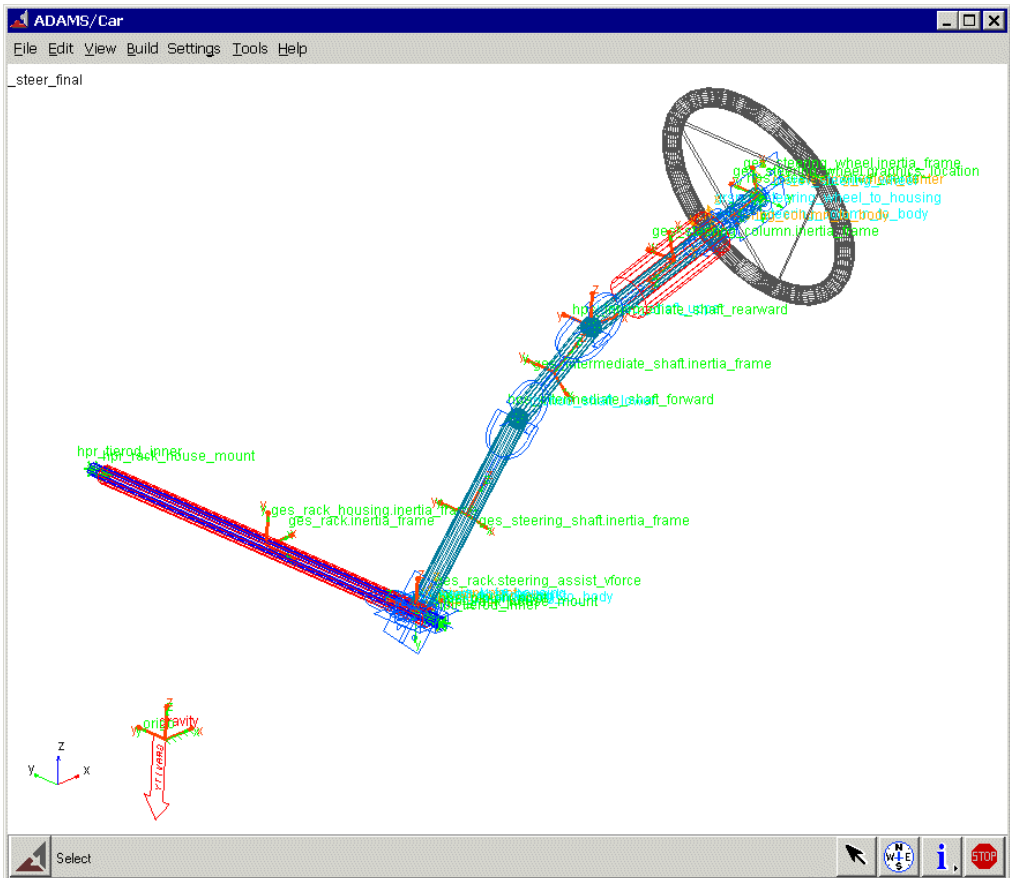
- 1 Create two hooke (universal) joints: one between the steering column and the intermediate shaft, and one between the intermediate shaft and the steering shaft.
- 2 Create a reduction gear to constrain the revolute joint for the steering wheel to the cylindrical joint of the steering column.
- 3 Create a revolute joint between the end of the steering shaft and the rack housing.
- 4 Create a reduction gear that constrains the rotational motion of the steering shaft to the translational motion of the rack.
- 5 Create a translational joint between the rack and the rack housing.
- 6 Create a mount part that will attach to the body.

# Workshop 13—Exploring and Completing Templates...

- 7 Open the MacPherson template created earlier, and check the mount parts and mount communicators at the tierods. The mount input communicator tierod\_to\_steering in the MacPherson template requires that a mount output communicator named tierod\_to\_steering, which outputs the rack part, be defined in the template steer\_training.
- 8 The steering column is already attached to a mount part named steering\_column\_to\_body. Edit a body template in [Workshop 15—Full-Vehicle Assembly](#) on page 205 to make sure these communicators match properly.

Your template should look like steer\_final.tpl, which your instructor will have, and will be used in the final full-vehicle workshop.

- 9 If time allows, add location communicators to your template. See knowledge base article 9184, [Position one template \(part\) relative to another template](#).



# Workshop 13—Exploring and Completing Templates...

---

## Tips for exploring templates

**Note:** Because you perform many steps in this section in the Database Navigator, make sure you have the Database Navigator displayed (**Tools -> Database Navigator**).

### Investigating model topology

#### To list parts and connections:

- To see parts and connections, set the option menu at the bottom of the Database Navigator to **Bodies, Constraints, or Forces**.
- Set the option menu at the top to the type of information you want to see.

#### To view the topology of parts:

- 1 From the option menu at the top of the dialog box, select **Topology by Parts** or **Topology by Connections**.
- 2 From the tree list or view window, select an object.  
The topology of the object appears in the text box to the right.

#### To graphically view the topology of parts:

- 1 From the option menu at the top of the dialog box, select **Graphical Topology**.
- 2 From the tree list or view window, select an object.  
A graphical display of the object's topology appears in the text box to the right.

# Workshop 13—Exploring and Completing Templates...

---

## Viewing the associativity of objects

You can use the Database Navigator to display the objects that a selected object uses. For example, you can select a joint in the tree list to show the I and J markers that the joint uses. You can also select to view the objects that use the selected object.

### To view the associativity of objects:

- 1 From the option menu at the top of the dialog box, select **Associativity**.
- 2 Set the associativity:
  - ◆ To show the objects that the selected object uses, select **Uses**.
  - ◆ To show the objects that use the selected object, select **Is Used By**.
- 3 From the tree list or view window, select an object.

The objects associated with the selected object appear in the text box to the right.

### To set up automatic navigation of the objects:

- To see what objects are dependent on a certain object, select **Auto Navigate**.

### To save the current associativity information to a file:

- Select **Save to File**.

# Workshop 13—Exploring and Completing Templates...

---

## Viewing object information through the Database Navigator

You can use the Database Navigator just as you would the Information window to display information about the selected object.

### To display object information:

- 1 Set the option menu at the top of the Database Navigator to **Information**.
- 2 From the tree list or view window, select an object.

The information about the object appears in the text box to the right.

### To save the information to a file:

- Select **Save to File**.

### To list the type of communicators:

- 1 From the **Build** menu, point to **Communicator**, and then select **Info**.
- 2 Set **Type** to **Input**.
- 3 Set **Entity** to **All**.
- 4 Select **OK**.

### To see what is dependent on a communicator:

- 1 From the **Tools** menu, select **Database Navigator**.
- 2 In the **Filter** text box, enter **ci\***.
- 3 Set the option menu at the top of the dialog box to **Associativity**.
- 4 To get the relevant information, select **Uses** and **Is Used By**.

# Workshop 13—Exploring and Completing Templates...

## Location and orientation parametrics

To find out how the parametric dependencies are set up, you can regard two entities as starting points: hardpoints and input communicators.

Hardpoints have no dependencies. Therefore, you can regard them as starting points for parametrics. You can parameterize construction frames and other components to hardpoints.

You use parameter variables to parameterize the location and orientation of construction frames. For more information, see the guide, *Learning ADAMS/View Basics*.

### To see what is dependent on a hardpoint:

- 1 In the **Filter** text box, enter **hp\***.
- 2 Select a hardpoint.
- 3 Select **Apply**.

The Information window appears, displaying hardpoint location values.

- 4 In the Database Navigator, set the option menu at the top of the dialog box to **Associativity** to see a list of dependents. Also, you can select **Highlight** to see the hardpoint position.
- 5 You can right-click each construction frame and select **Info** or **Modify**. The information displayed includes information for the parametric locations in the template. For construction frames, you look both at the expression to see how the construction frame is parametric to other construction frames and hardpoints, and what is dependent on the construction frame.

### To see what is dependent on construction frames:

- 1 In the **Filter** text box, enter **cf\***.
- 2 Set the option menu at the top of the dialog box to **Associativity**.

### To list dependencies for parameter variables:

- 1 In the **Filter** text box, enter **pv\***.
- 2 To see the parameter variables, set the option menu at the top of the dialog box to **Information**.
- 3 To see a list of dependents, set the option menu at the top of the dialog box to **Associativity**.



Other applications offered by Mechanical Dynamics are compatible with ADAMS/Car to provide additional analysis capabilities. The following describes how these applications fit in and affect the vehicle simulations in ADAMS/Car. For more information, refer to the documentation for each application.

### What's in this module:

- Conceptual Suspension Module and Driveline, 194
- Linear and Controls, 196
- Insight and Hydraulics, 197
- Vibration and Durability, 198

## ADAMS/Conceptual Suspension Module (CSM)

CSM is a method of implementing functional suspension behavior in ADAMS/Car through predefined trajectories followed by the wheel carrier during suspension travel (vertical and steer) and external forces. You benefit from using CSM because it concentrates only on final effects (wheel carrier position and orientation) of the suspension layout, regardless of the way in which they have been mechanically obtained. It allows you to have a reduced 14 degree-of-freedom vehicle model that can be quickly assembled, and includes all the primary nonlinear elasto-kinematic effects of a multibody suspension model. In addition, conceptual suspension modeling allows you to share suspension characteristics files (.scf) with others, such as suppliers, without worrying about confidentiality.

## ADAMS/Driveline

ADAMS/Driveline is a new ADAMS module that you can use to quickly build and test functional virtual prototypes of complete drivelines or driveline components. ADAMS/Driveline is implemented as an add-on module to ADAMS/Car, has the same look and feel as ADAMS/Car, and it allows you to perform the same kinds of analyses.

You can use the virtual prototypes created with ADAMS/Driveline for performance analyses, component fatigue life estimation, and NVH characteristics. You can also import those virtual prototypes into ADAMS/Car to study full-vehicle dynamics with a driveline included.

Similar to other automotive applications, such as ADAMS/Car and ADAMS/Engine, ADAMS/Driveline has two operational modes:

- Standard Interface – Allows designers and analysts to build up driveline designs from a database of available templates, fill in the design-specific numerical values, and run tests.
- Template Builder – Allows experienced users to modify templates, or create new ones, to accommodate specific corporate needs.

# Conceptual Suspension Module and Driveline...

---

Application areas for ADAMS/Driveline include:

- Conceptual tool for driveline layout
- Full-vehicle handling analysis in front-wheel drive, rear-wheel drive, and all-wheel drive configurations
- Torque transfer and torque distribution studies
- Vibration analysis at the system level and the component level
- Component fatigue life prediction
- Control system design and verification
- Driveline packaging studies
- Studies of secondary motion at high operating speeds
- Studies of the influence of flexible components

ADAMS/Driveline template libraries provide a wide range of components, including clutches, flexible shafts, synchronized gear pairs, bearings, viscous couplings, differentials, backlash, and gears.

ADAMS/Driveline catalog of tests and analyses includes split m, uphill and downhill driving, impulse, step, and ramp torques, tip-in tip-out, gear shift, rock cycle, and clutch misuse.

## ADAMS/Linear

ADAMS/Linear's key function is to linearize nonlinear ADAMS equations. The resulting set of equations enables you to calculate the natural frequencies (eigenvalues) and mode shapes (eigenvectors) associated with your mechanical design. ADAMS/Linear also helps bridge the gap between control systems design and ADAMS mechanical simulation capabilities by allowing you to generate state space matrices (the plant model), which relate the measured outputs of the system to the controlled input.

## ADAMS/Controls

Control algorithms can now be used in the vehicle model. For example, including automatic controls in the suspension and handling design of a vehicle makes it possible to avoid an obstacle on wet pavements. The control system designer can tune the controller for the particular vehicle design on the computer, with confidence that the ADAMS model provides accurate results. ADAMS/Controls allows the designer to see the effect of each change on the computer, even comparing control strategies that would be too expensive or time-consuming to test on a physical prototype. This interface allows you to model control algorithms in Matlab, MatrixX, or Easy5.

## ADAMS/Insight

When using ADAMS/Insight together with ADAMS/Car, you can plan and run a series of experiments for measuring the performance of your suspension or vehicle. ADAMS/Insight will help you to understand your designs better by distinguishing between key and insignificant design parameters. In addition, you will be able to see the impact of design decisions on a broad range of customer requirements, as well as improving design robustness by considering manufacturing variations up front. To learn how to use ADAMS/Car with ADAMS/Insight, see the guide, *Using ADAMS/Insight with ADAMS/Car*.

## ADAMS/Hydraulics

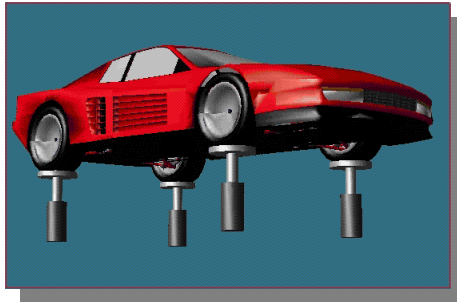
This module allows you to smoothly integrate system-level motion simulation and hydraulic system design. For example, auto companies need to know the effects of hydraulic system failures on vehicle performance and safety – especially since high ratings in these areas are key competitive advantages. Example: How is a sudden change in power steering authority sensed by the driver while negotiating a curve at top speed? Answering this question requires failure analyses using simulations that include not just mechanical systems, but also hydraulic systems and other control system designs.

## ADAMS/Vibration (New in 11.0)

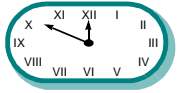
ADAMS/Vibration is a new plug-in module, which allows you to analyze system modes including attachment characteristics, include effects of hydraulics and controls on system behavior, and analyze vibratory behavior in different configurations. Moreover, it allows you to study forced vibrations within your ADAMS models using frequency domain analysis. For example, you can drive an ADAMS model of an automobile over a bumpy road and measure its response. Both inputs and outputs are described as vibrations in the frequency domain.

## ADAMS/Durability (New in 11.0)

ADAMS/Durability allows you to perform fatigue analyses using a virtual test rig, by providing an interface to MTS and nCode products. The virtual test rig loads will be based on empirical data to provide better virtual test results and to allow better corroboration when the actual testing is performed.



# Workshop 14—Using ADAMS/Linear with ADAMS/Car



This workshop takes about one hour to complete.

In this workshop, you use ADAMS/Linear to analyze the mode shapes of a full-vehicle about an operating point.

ADAMS/Linear will linearize your vehicle model about an operating point, and return the eigenvalues and eigenvectors to help validate your model or investigate excited frequencies in your vehicle. For example, a sweep of animations of the modes will help indicate that parts are connected properly. To get the eigenvalues and eigenvectors, use an .acf file in ADAMS/Car to submit the LINEAR commands. ADAMS/View can perform this analysis from the toolbar, because it has an interface for this function. Because ADAMS/Car doesn't have this interface, the most straightforward method is to use an .acf file with the external ADAMS/Solver. The first part of the workshop will generate this .acf file.

# Workshop 14—Using ADAMS/Linear with ADAMS/Car...

## Generating the .acf file

### To generate the .acf file:

- 1 In ADAMS/Car, open `MDI_Demo_Vehicle.asy`.
- 2 From the **Simulate** menu, point to **Full-Vehicle Analysis**, point to **Straight-Line Behavior**, and then select **Acceleration**.
- 3 Fill in the acceleration simulation dialog box with the following parameters:

- ◆ **Output Prefix:** `linear_example`
- ◆ **End Time:** `5`
- ◆ **Number Of Steps:** `100`
- ◆ **Mode of Simulation:** `files_only`

Specify `files_only`, because you do not need to run the simulation yet. This step is used to simply set up the vehicle model and create the `.adm`, `.acf`, and `.dcf` files. Here, ADAMS/Car will read in the property files and store the relevant data in the `.adm` file, because if you submit an analysis through the interface in ADAMS/Car, the property files are loaded for you automatically. If you want to load the property files manually, you can do this with View Commands (`acar analysis read property_files`).

- ◆ **Road Data File:** `<shared>\roads.tbl\mdi_2d_flat.rdf`
- ◆ **Initial Velocity:** `10 km/hr`
- ◆ **Start Time:** `1`
- ◆ **Open-Loop Throttle**
- ◆ **Throttle Ramp:** `10`
- ◆ **Gear Position:** `2`
- ◆ **Steering Input:** `locked`



# Workshop 14—Using ADAMS/Linear with ADAMS/Car...

## 4 Select OK.

ADAMS/Car displays the following message:

Reading in property files...

Reading of property files completed.

Setting up the vehicle assembly for Driving Machine maneuver...

Setup of vehicle assembly completed.

Writing assembly information to ADAMS/Solver dataset...

ADAMS/Solver files written successfully.

ADAMS/Car writes the files to your current working directory. To check where ADAMS/Car saved the files, go to File --> Select Directory)

To set up your .acf file, you simply insert the LINEAR solver command after your analysis, specifying whatever options you want (see the ADAMS/Solver documentation for LINEAR). ADAMS will linearize about the operating point (the states) that it sees at the end of the analysis. For example, here is the .acf file you just created:

```
file/model=linear_example_accel
output/nosep
control/function=user(906,2,29,33,22,3,23,1)
!
stop
```

## 5 Edit the linear\_example\_accel.acf file to include an eigensolution.

Your .acf file should look like:

```
file/model=linear_example_accel
output/nosep
control/function=user(906,2,29,33,22,3,23,1)
  linear/eigensol
!
  STOP
!
```

You are almost ready to run the linear analysis. First, you must make sure you save the proper output file. To save your mode shape results, you have to turn on the results file (.res), which is usually turned off for ADAMS/Car. To produce the results file, you can do this in the Output Files dialog box by selecting Results File Contents (that is, Settings --> Solver --> Output Files). In the window that appears, make sure that the Linear box is checked.

**Note:** You must do this before you export the .adm file.

Alternatively, you can simply put the RESULTS/ statement at the end of your .adm file.

# Workshop 14—Using ADAMS/Linear with ADAMS/Car...

- 6 Edit the .adm file by putting the RESULTS/ statement at the end of the .adm file.

```
!----- ANALYSIS SETTINGS -----  
!  
INTEGRATOR/  
, GSTIFF  
, ERROR = 0.01  
!  
OUTPUT/  
, REQSAVE  
, GRSAVE  
, NOPRINT  
!  
RESULTS/  
END
```

## Running the model with the external solver

### To run the model with the external solver:

- 1 Enter your alias to start ADAMS at your command line (for example, **adams110**, **adams11**).

**Note:** Aliases can vary. The alias simply points to the mdi.bat file, so if you find no alias, run the mdi.bat file found in \$install\_dir/common/mdi.bat.

- 2 To see the ADAMS/Car options, enter **acar**.
- 3 To run the ADAMS/Car solver, enter **ru-solver**.
- 4 To enter your solver commands, enter **linear\_example\_accel.acf**.

**Note:** Make sure that this file and all others are in the directory in which you are running ADAMS.

The simulation should start. It will produce output files, including the results file which has the eigensolution output in it. Alternatively, you can submit the simulation directly by entering:

- ◆ UNIX: **adams11 -c acar ru-solver linear\_example\_accel.acf exit**
- ◆ NT: **adams11 acar ru-solver linear\_example\_accel.acf exit**

## Reviewing mode shapes in ADAMS/PostProcessor

### To review mode shapes:

- 1 Open ADAMS/Postprocessor.
- 2 From the **File** menu, point to **Import**, and then select **Analysis Files**.
- 3 In the **File Name** text box, enter one of the three files, **linear\_example\_accel.gra**, **linear\_example\_accel.res**, or **linear\_example\_accel.req**. Entering one will pick up all three, based on the root name.  
  
You should see your vehicle model loaded into ADAMS/PostProcessor.
- 4 Right-click in the empty space around the model and select **Load mode shape animation**.
- 5 When you get a message that the animation will be deleted, select **OK**.
- 6 Review the mode shapes by entering the respective mode shape number of interest in the **Mode Number** text box. To review the eigenvalues corresponding to the mode shapes, select **Table of Eigenvalues**. For example, select mode 160, and use the play tool to animate the mode shape. You should see in the eigenvalue table that this mode has real and imaginary values for the eigenvalue:

EIGEN VALUES (Time = 5.0)				
FREQUENCY UNITS: (hz)				
MODE	UNDAMPED NATURAL	DAMPING		
NUMBER	FREQUENCY	RATIO	REAL	IMAGINARY
160	8.826190E+000	6.391413E-001	-5.641182E+000	+/- 6.788129E+000

# Workshop 14—Using ADAMS/Linear with ADAMS/Car...

---

## Background information about ADAMS/Solver

### Files in ADAMS/Solver

- ADAMS/Solver dataset files (.adm) - Statements define an element of a model such as a part, constraint, force, and so on.
- ADAMS/Solver command files (.acf) - Commands define an action that needs to be taken during a simulation.

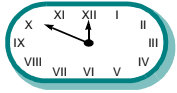
### Simulations in stand-alone ADAMS/Solver

- Interactive
- Not scripted - Enter commands one by one
- Scripted - Use an ADAMS/Solver command file (.acf)
- Batch - Run multiple jobs in the background using an ADAMS/Solver command file (acf).

**Note:** ADAMS/Solver command files must start with the name of the model to be analyzed and must end with a STOP command.

You can run simulations externally in ADAMS/Solver from within ADAMS/View.

# WORKSHOP 15—FULL-VEHICLE ASSEMBLY



This workshop takes about six hours to complete.

This workshop is intended to combine the knowledge you have attained throughout the course and challenge your knowledge of different topics. The major steps for this workshop are described, but the exact steps are omitted. If you have questions, please refer to previous sections or ask the instructor.

At this point, you should have the following templates done:

- MacPherson suspension (be sure to use the one you created rather than the one in the shared database, because these have topological differences)
- Rack and pinion
- Wheel

You should also have access to `_double_wishbone.tpl`, which is located in the shared database.

Before you build the assembly, you must create the body. You then attach the front and rear suspensions, together with the steering system, to the body. You also create appropriate communicators.

## Creating the body

### To create the body:

- In Template Builder, open the template **body\_training.tpl**, which you will get from the instructor. This template has one rigid body that will act as the chassis.

# Workshop 15—Full-Vehicle Assembly...

## Creating and testing communicators

### To create and test communicators:

- 1 Make sure the front suspension hooks up to the body. For the MacPherson suspension, you have the following input communicators:
  - `ci[lr]_strut_to_body`            `mount`
  - `ci[lr]_subframe_to_body`        `mount`
  - `ci[lr]_tierod_to_steering`       `mount`
- 2 The communicators of interest are **strut\_to\_body** and **subframe\_to\_body**, which are mount communicators. You must convey to the front suspension what part the upper strut should attach to. In this case it's the body, since that is the only part in the chassis template.
- 3 Create a new output communicator of type mount, that will match the `ci[lr]_strut_to_body`, by either entering the same name or entering the same matching name.
- 4 Edit the communicator for **strut\_to\_body** to have a matching name for **subframe\_to\_body**.
- 5 Test the communicators. Remember that you must have the templates open to be able to test them.
- 6 Set up the steering and body template so that they will attach to each other. That is, make sure that the steering column housing and rack housing are attached to the body. The rack housing should really be attached to the suspension subframe, but since you don't have a subframe for the MacPherson suspension, use the body instead.
- 7 Display the body template and create an output communicator of type mount single with the name **steering\_column\_to\_body** or with a matching name of the corresponding input communicator in the steering template.
- 8 Modify the same single output communicator to include the matching name **rackhousing\_to\_body**.
- 9 Test the communicators.
- 10 Make sure that the rear suspension hooks up to the body. Look in the ADAMS/Car documentation: **Help -> ADAMS/Car Guides -> Building Templates in ADAMS/Car -> Template Descriptions -> Double-Wishbone Suspension**.
- 11 Make sure that the strut attaches to the body. This is the same communicator you created for the front suspension, so you don't have to create it again. It should pass the information to both the front and the rear mount parts at the same time.

## Workshop 15—Full-Vehicle Assembly...

- 12 Open the rear suspension template and go to **Build -> Communicator -> Info**, select **Input** and **Many**, and then select **mount**.
- 13 Select **OK**.

ADAMS/Car displays a list of all input communicators in the double-wishbone suspension template.
- 14 Modify the left/right body communicator in the body to pass information to the **uca\_to\_body** input communicator located in the double-wishbone suspension template.
- 15 You must modify the left/right body communicator to include **tierod\_to\_steering**. If you want this suspension to work as a steerable suspension, you must make sure it is attached to the steering system. To keep the wheels going straight, hook up the tierods to the body, so then the wheels won't turn.
- 16 Modify the single body output communicator to include the matching name **subframe\_to\_body** for the double-wishbone suspension.
- 17 These suspensions also need to pass on the information to what part the wheels should attach to. You must edit output communicators, in the suspensions, named **suspension\_mount** to include the matching name **wheel\_mount**.
- 18 Remember to check the test rig, **SDI\_testrig**. There is one mount input communicator that expects information from the body.
- 19 Test all the communicators for all the templates that you will use for the full vehicle.
- 20 When testing these communicators, you must specify what the templates' minor roles will be when you will build the subsystems. Assign minor roles to templates, as shown next:

Template:	Minor role:
._MDI_SDI_TESTRIG	any
._new_tire	front
._new_tire	rear
._body_training	any
._double_wishbone	rear
._steer_training	front
._macpherson	front

# Workshop 15—Full-Vehicle Assembly...

- 21** The following is a listing of the most important mount and location communicators for this workshop. You may use this as a guide if you are having difficulties or would like to check your work.

Listing of input communicators in `_body_training`:

Communicator Name:                      Class:                      From Minor Role:                      Matching Name:

(the body template shouldn't need input communicators for this exercise)

'`_body_training`' contains:

0 input communicators of type 'location'

0 input communicators of type 'mount'

Listing of output communicators in '`_body_training`'

Communicator Name:                      Class:                      To Minor Role:                      Matching Name:

(need an output communicator of type mount to output the body part)

symmetry, matching names needed:

lr, `strut_to_body` (macpherson, dwb)

lr, `subframe_to_body` (macpherson)

s, `subframe_to_body` (dwb)

lr, `uca_to_body` (dwb)

lr, `tripot_to_differential` (dwb) (unless have powertrain template)

s, `steering_column_to_body` (rack and pinion)

s, `rackhousing_to_body` (rack and pinion)

s, `body_subsystem` (SDI\_testrig)

'`_body_training`' contains:

0 output communicators of type 'location'

0 output communicators of type 'mount'

Listing of input communicators in '`_new_tire`'

Communicator Name:                      Class:                      From Minor Role:                      Matching Name:

1. ci[`lr_wheel_center`]                      location                      inherit                      `wheel_center`

2. ci[`lr_wheel_mount`]                      mount                      inherit                      `wheel_mount`

Should have matching name of "suspension\_mount" for `_macpherson` template.

'`_new_tire`' contains:

2 input communicators of type 'location'

2 input communicators of type 'mount'

Listing of output communicators in '`_new_tire`'



## Workshop 15—Full-Vehicle Assembly...

-----  
Communicator Name:                      Class:                      To Minor Role:                      Matching Name:

'\_new\_tire' contains:

0 output communicators of type 'location'

0 output communicators of type 'mount'

Listing of input communicators in '\_steer\_final'

-----

Communicator Name:                      Class:                      From Minor Role:                      Matching Name:

1. cis\_rackhousing\_to\_body                      mount                      inherit                      rackhousing\_to\_body

Should connect to the body template. Alternatively, you could change this to connect the rack housing to the suspension subframe. However, the macpherson template we created does not have a subframe, so you would have to use another template to do this.

2. cis\_steering\_column\_to\_body                      mount                      inherit

steering\_column\_to\_body

Should connect to the body template.

'\_steer\_final' contains:

0 input communicators of type 'location'

2 input communicators of type 'mount'

Listing of output communicators in '\_steer\_final'

-----

Communicator Name:                      Class:                      To Minor Role:                      Matching Name:

1. co[lr]\_tierod\_to\_steering                      mount                      inherit                      tierod\_to\_steering

Should connect to the macpherson template. Outputs the "rack" part.

(could create an output communicator of type location to locate the tierod to the end of the rack; see KB 9182 for example of this)

Listing of input communicators in '\_macpherson'

-----

Communicator Name:                      Class:                      From Minor Role:                      Matching Name:

1. ci[lr]\_strut\_to\_body                      mount                      inherit                      strut\_to\_body

Should connect to the body template.

2. ci[lr]\_subframe\_to\_body                      mount                      inherit                      subframe\_to\_body

Should connect to the body template.

3. ci[lr]\_tierod\_to\_steering                      mount                      inherit                      tierod\_to\_steering

Should connect to the rack and pinion template.

(could create input communicator of type location to locate the tierod to the end of the rack; see KB 9182 for example of this)

'\_my\_mac' contains:

0 input communicators of type 'location'

6 input communicators of type 'mount'

# Workshop 15—Full-Vehicle Assembly...

Listing of output communicators in \_macpherson:

-----

Communicator Name:	Class:	To Minor Role:	Matching Name:
1. co[lr]_wheel_center	location	inherit	wheel_center
Outputs the location to the wheel template.			
2. co[lr]_suspension_mount	mount	inherit	suspension_mount
Should have matching name of "wheel_mount" for wheel template. Outputs the "hub" part.			

'\_my\_mac' contains:  
2 output communicators of type 'location'  
2 output communicators of type 'mount'

Listing of input communicators in \_double\_wishbone:

-----

Communicator Name:	Class:	From Minor Role:	Matching Name:
1. ci[lr]_ARB_pickup	location	inherit	arb_pickup
Don't worry about this one since we don't have an ARB.			
2. ci[lr]_strut_to_body	mount	inherit	strut_to_body
Body template needs to have an output communicator with itself as the output part.			
3. ci[lr]_tierod_to_steering	mount	inherit	tierod_to_steering
Body template needs to have an output communicator with itself as the output part in order to fix the steering in the rear.			
4. ci[lr]_tripot_to_differential	mount	inherit	tripot_to_differential
This should be connected to the body, as above. However, when adding a powertrain, the body communicator should be removed.			
5. ci[lr]_uca_to_body	mount	inherit	uca_to_body
Body needs output communicator with itself as the output part.			
6. cis_subframe_to_body	mount	inherit	subframe_to_body
Body needs output communicator with itself as the output part.			

'\_double\_wishbone' contains:  
2 input communicators of type 'location'  
9 input communicators of type 'mount'

# Workshop 15—Full-Vehicle Assembly...

Listing of output communicators in `_double_wishbone`:

-----

Communicator Name:	Class:	To Minor Role:	Matching Name:
--------------------	--------	----------------	----------------

1. <code>co[lr]_tripot_to_differential</code>	<code>location</code>	<code>inherit</code>	<code>tripot_to_differential</code>
---	-----------------------	----------------------	-------------------------------------

You don't need to worry about this one. This will tell the powertrain where to locate itself if you use one.

2. <code>co[lr]_wheel_center</code>	<code>location</code>	<code>inherit</code>	<code>wheel_center</code>
-------------------------------------	-----------------------	----------------------	---------------------------

This feeds the wheel center communicator in the wheel template.

3. <code>co[lr]_arb_bushing_mount</code>	<code>mount</code>	<code>inherit</code>	<code>arb_bushing_mount</code>
--	--------------------	----------------------	--------------------------------

You don't need to worry about this one since we don't have an ARB.

4. <code>co[lr]_droplink_to_suspension</code>	<code>mount</code>	<code>inherit</code>	<code>droplink_to_suspension</code>
---	--------------------	----------------------	-------------------------------------

You don't need to worry about this one since we don't have an ARB.

5. <code>co[lr]_suspension_mount</code>	<code>mount</code>	<code>inherit</code>	<code>suspension_mount</code>
---	--------------------	----------------------	-------------------------------

Should have matching name of "wheel\_mount" for wheel template. Outputs the "spindle" part.

6. <code>co[lr]_suspension_upright</code>	<code>mount</code>	<code>inherit</code>	<code>suspension_upright</code>
---	--------------------	----------------------	---------------------------------

Don't worry about this one. Outputs the "upright" part.

7. <code>cos_engine_to_subframe</code>	<code>mount</code>	<code>inherit</code>	<code>engine_to_subframe</code>
--	--------------------	----------------------	---------------------------------

Don't worry about this one.

8. <code>cos_rack_housing_to_suspension_subframe</code>	<code>mount</code>	<code>inherit</code>	
---	--------------------	----------------------	--

`rack_housing_to_suspension_subframe`

You may want to hook this up to your rack housing in your rack and pinion template if you want to hook the rack housing to the subframe. Otherwise, just connect the rack housing to the body, which is probably simpler in this case. Outputs the "subframe" part.

'`_double_wishbone`' contains:

4 output communicators of type 'location'

10 output communicators of type 'mount'

# Workshop 15—Full-Vehicle Assembly...

Listing of input communicators in \_\_MDI\_SDI\_TESTRIG:

-----			
Communicator Name:	Class:	From Minor Role:	Matching Name:
cis_body_subsystem	mount	inherit	body_subsystem
'__MDI_SDI_TESTRIG' contains:			
0 input communicators of type 'location'			
1 input communicator of type 'mount'			

- 22 Create new subsystems for all templates.
- 23 Create a new full-vehicle assembly with the new subsystems you just made. Make sure that none of the critical communicators appear in the warning message about unmatched communicators.
- 24 Adjust the different subsystems so they appear in the right location relative to each other. Shift the front and the rear suspensions by this amount:

Front suspension	aft 250 mm
	up 230
Rear suspension	aft 2800 mm
	up 37.3 mm
- 25 View only the double-wishbone suspension.
- 26 From the **Adjust** menu, select **Driveline Activity**.
- 27 Set **Current Mode** to **Inactive**.
- 28 Run a full-vehicle steering analysis.

This appendix provides a basic outline of how you can use ADAMS/Car to analyze your vehicle.

### What's in this appendix:

- Types of Analyses, 214
- Gather Data for the Model, 215
- Packaging Analysis on Suspension, 216
- Kinematic Analysis on Suspension, 217
- Suspension-Compliance Analysis, 219
- Static-Loading Durability Analysis, 220
- Dynamic-Loading Durability Analysis, 222
- Front Suspension Analyses, 223
- Full-Vehicle Design and Analysis, 224

# Types of Analyses

---

**Below is a list of analyses that you can perform ADAMS/Car:**

- Packaging analysis on suspension
- Kinematic analysis on suspension
- Suspension-compliance analysis on suspension
- Static-loading durability analysis on suspension
- Dynamic-loading durability analysis on suspension
- Full-vehicle design and analysis

**In the following pages, a guide is given to perform these analyses. This guide only lists a subset of the analyses available in ADAMS/Car. Moreover, some additional modules may be necessary for the analyses listed (for example, ADAMS/Durability). For more information on a particular analysis, see the documentation.**

## Gather Data for the Model

---

Collect the data necessary to generate an ADAMS model of the proposed suspension concept. After you've collected the data from various sources (CAD data for geometry, bushing data from internal testing facilities or from bushing supplier, shock data, and so on), you can create an ADAMS/Car model.

## Packaging Analysis on Suspension

---

Once you've created the ADAMS model, you put the virtual suspension on a virtual test fixture (standard part of ADAMS/Car) and run through a series of events to examine packaging and interference issues. The goal of this analysis is to show that parts do not collide during jounce and roll travel.

Also, if body geometry is available, be able to demonstrate that tire and wheel well clearances conform to corporate standards. The goal of this phase of the analysis is to give a cursory check of the part collisions within the ADAMS/Car environment.

Further investigations are possible by bringing the ADAMS results for the wheel envelope during maximum jounce/rebound/roll travel into your CAD package. You can use the solid geometry of the wheel envelope in your CAD package to easily find clearance issues, and to better visualize the total wheel envelope needed by your suspension concept.



# Kinematic Analysis on Suspension

---

After you've analyzed the suspension packaging issues, put the virtual suspension on a virtual test fixture and run through a series of events to understand the kinematic properties of the suspension. Two analyses help you understand the suspension kinematics: parallel wheel travel (wheels moving vertically in phase) and roll travel (wheel moving vertically out of phase).

**1. The parallel wheel travel analysis (also called the ride travel event) examines the following suspension metrics:**

- Toe (also called bump steer)
- Caster
- Camber
- Longitudinal recession
- Lateral recession

Wheel rate (vertical force from suspension springs versus amount of suspension vertical deflection)

**2. The roll travel analysis examines the following suspension metrics:**

- Roll steer (degrees of toe per degree of suspension roll)
- Roll stiffness

## Kinematic Analysis on Suspension...

---

The goal of the kinematic analysis is to tune the geometry of the suspension to attain satisfactory kinematic behavior. If kinematic issues arise, design recommendations can then be made to the location of suspension joints and bushings, the lengths of the control arms, and other geometric properties that affect the kinematics of the suspension.

Also, suspension spring properties can be examined to be sure that the overall vehicle requirements will be met for suspension springs during ride and roll. Use of additional suspension components such as an anti-roll bar can be examined, including recommendations about the sizing for the anti-roll bar.

# Suspension-Compliance Analysis

---

After you analyze the suspension geometry and the design shows good kinematic behavior, you can examine suspension compliance. The virtual suspension model will be placed on the suspension test rig and run through the compliance analysis (for example, static loading). The following metrics are generated with this analysis:

- Lateral force versus toe (for both parallel and opposing lateral force)
- Lateral force versus camber (for both parallel and opposing lateral force)
- Lateral force versus lateral displacement (for both parallel and opposing lateral force)
- Longitudinal force versus toe (braking and acceleration forces)
- Longitudinal force versus longitudinal displacement (braking and acceleration forces)
- Aligning torque versus toe (parallel and opposing torques)

The goal of the suspension compliance analysis is to tune the suspension bushings such that adequate suspension compliance is attained. Note that ball joints are not infinitely stiff, and they do factor in to the suspension performance. Thus, it may be a good idea to replace idealized ball joints in your model with bushing representations.

# Static-Loading Durability Analysis

---

The next step in analyzing the suspension is to apply static loadcases to the wheels in ADAMS/Car and examine the resulting loads of the suspension elements (suspension bushings, suspension springs, and so on). This contributes to a better understanding of the durability of the suspension.

Typically, a set of requirements for static loads are used which take the form of a worst case loading condition that a suspension must withstand. These loading conditions (or loadcases) take the form of number of g's of loading. For example, a suspension requirement might be that it must withstand 3 g's of vertical load, 2 g's of longitudinal load, and 1 g of lateral load. Such a loading condition is often referred to as a "321 g" loadcase.

The use of "g's" describes the total vehicle weight, divided by the front to rear load distribution. In this way, a loadcase requirement in "g's" of load can be used across different vehicles of various sizes and weights.

The goal of the loadcase analysis is to give the design and FEA analyst a report which shows the worst case loading on each of the suspension components (control arm, bushings, spring, and so on). This data can then be fed into a FEA model of the component (bushing, control arm, spring, and so on) to show that the component will withstand a given amount of static loading. A structured report will be generated through the use of the loadcase postprocessor which will show the amount of loading on the suspension elements, and using this report the loading data can be imported into the FEA software (NASTRAN, ANSYS, ABAQUS, and so on).

## Static-Loading Durability Analysis...

---

It is rare that the design engineer can have access to this level of data early in the program, so the static loadcase analysis provides the first insight to real-world loading conditions for the parts. This method is coarse and does not provide the final answer to the durability requirements of the suspension components, but early in the design and analysis, this information is extremely valuable.

## Dynamic-Loading Durability Analysis

---

After the static durability analysis is completed, a more intensive investigation into the rubber mounts begins. In this phase, a road profile will be assumed for the suspension (for example, a pothole, or random white noise road input) and utilized on a virtual four-post shaker event within ADAMS/Car. The ADAMS dynamic loading histories of the mounts will then be imported into FEA software for final structural analysis.

# Front Suspension Analyses

---

Besides the analyses already listed, specific to the front suspension, steering system metrics will also be examined to understand the overall steering ratio, steering linearity, Ackerman, and steering column universal joint phasing (ensure that the steering feel is not lumpy).

# Full-Vehicle Design and Analysis

---

Once the rear suspension, front suspension, and steering system subsystems have been modeled in ADAMS/Car, the next stage of the Functional Digital Car process is to analyze the full-vehicle behavior. Because the various suspension subsystem models have been developed in the earlier stages of the plan, it will be a simple matter to assemble these subsystems into a full-vehicle ADAMS/Car model.

This ADAMS/Car model will be driven on a set of virtual test tracks to understand various full vehicle metrics. Example test analyses would be:

- Static vehicle trim analysis to trim the vehicle to a particular ride height
- Double lane change (Moose Test) to examine rollover propensity in an accident-avoidance maneuver
- Constant radius to examine understeer/oversteer behavior of the vehicle and generate an understeer budget report
- Brake drift to examine amount of lateral drift during a moderate-braking maneuver
- Brake in a turn to examine dynamic braking and turning stability issues

## Example test events:

- Step steer to examine lateral acceleration and yaw rate overshoot during highly dynamic maneuver
- Frequency response to examine vehicle dynamics metrics in the frequency domain
- Dynamic durability to determine the behavior of the vehicle as it drives over potholes, bumps, and other three dimensional road obstacles



This tutorial teaches you how to work with custom ADAMS/Car test rigs, custom simulation procedures, and how to create a private ADAMS/Car binary.

You will work with an existing test rig created in ADAMS/Car Template Builder. This tutorial will overview the internal workings of the test rig, cover the steps used in making it, and then detail how to edit it and create a custom ADAMS/Car binary containing the test rig.

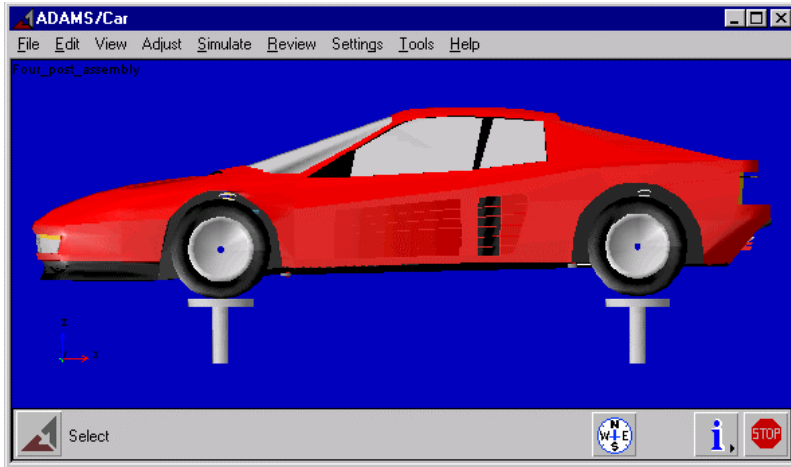
To go through this tutorial, you must have access to the six four-post test rig model, macro, and interface files: `acar_build.cmd`, `acme_3PostRig.cmd`, `acme_four_sim.cmd`, `macros_ana.cmd`, `mac_ana_ful_fou_sub.cmd`, and `acme_4PostRig.cmd`.

### What's in this appendix:

- [Introduction, 226](#)
- [Creating a test rig template in ADAMS/Car, 230](#)
- [Preparing to create a private binary, 239](#)
- [Creating an assembly using the test rig, 243](#)
- [Adding custom analysis procedures to ADAMS/Car, 247](#)

# Four-Post Vertical Excitation Test...

## Introduction



The objective of this exercise is to investigate the vertical dynamics of the full vehicle and its suspension systems. The test results can be postprocessed in the frequency domain to study the damped natural frequencies of various ride modes and their respective damping levels. Additional insight can also be gathered into the influences of the vehicle's vertical dynamics' effects on handling behavior by gaining a further understanding of system dynamic responses including:

- Front to rear modal balance
- Suspension to body transfer function gain and phase
- Suspension to tire transfer function gain and phase
- Tire contact patch vertical load variation

The test is conducted by assembling a standard full-vehicle model to a special four-post test rig. The test rig is simply defined by four parts representing the tire pads that support the vehicle. These tire pads are constrained to move in only the vertical direction and a displacement actuator (motion controller) controls their vertical motion. The only constraint between the pads and the vehicle's tires is the friction of the tire itself. Because the Delft tire model supports zero velocity tire friction, this is all that is required to constrain the vehicle during the dynamic portion of the simulation.

# Four-Post Vertical Excitation Test...

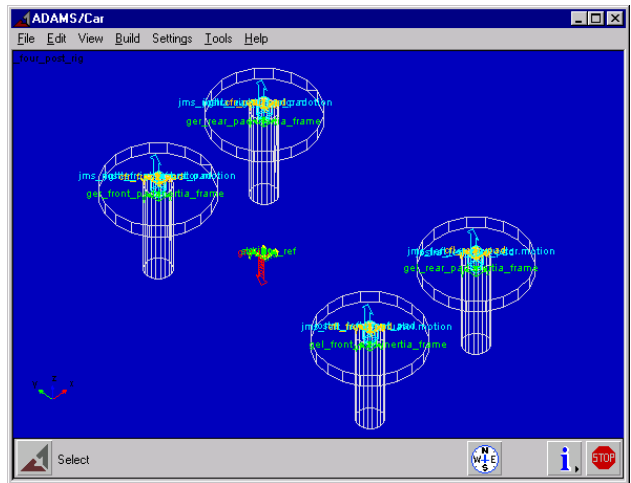
The vertical actuators are each controlled by an analytical function. The analytical functions are used to describe the displacement profile of the actuator in the time domain and they are limited to constant amplitude sinusoidal input that sweeps over a predetermined frequency range in a set amount of time. When using the analytical function control there exist four excitation modes described below:

- Heave: all tire pads move vertically in phase.
- Pitch: the front tire pads move 180 degrees out of phase with the rear tire pads.
- Roll: the left tire pads move 180 degrees out of phase with the right tire pads.
- Warp: the left-front and right-rear tire pads move 180 degrees out of phase with the right-front and left-rear pads.

## Test rig description

The four-post test rig was created in ADAMS/Car Template Builder and is named `__acme_4PostRig.tpl`. What follows is a brief description of how it works and how it relates to the standard ADAMS/Car simulation types.

The major role of the four-post test rig template is analysis and it contains four general parts. There are four tire pads and four actuators for each of the vertical translation joints on each pad. The location of all of the pads and their respective constraints, actuators, and so on, are parameterized in the ground plane (X and Y) to a wheel center location communicator that comes from the suspension systems. The vertical location is parameterized to the Z location of the `std_tire_ref` marker. The `std_tire_ref` marker has its Z height set automatically during the assembly process so that it represents the vehicles average tire contact patch height.



# Four-Post Vertical Excitation Test...

---

## User input requirements: simulation description

The analysis input parameters are grouped in two categories: one group of parameters is common to all analyses, while the other group consists of input specific to this four-post test. The four-post simulation input requirements are controlled by the user in order to define the boundary conditions of the desired vertical excitation test.

Always required:

- Output Prefix
- End Time
- Number of Steps
- Type of Analysis (Interactive, Background)
- Analysis Log File (Yes, No)

Four-post simulation input requirements:

- Peak Displacement
- Displacement Units (m, mm, inch, and so on)
- Frequency Range (Units hardcoded to Hz)
- Excitation Mode (Heave, Pitch, Roll, Warp)

## Assembly and simulation process requirements

The following steps outline what the test rig simulation process is doing internally after a simulation has been requested. The vehicle/test rig assembly process is similar regardless of the user input parameters outlined in the previous section of the document. However, the modifications made to the assembled model will vary depending on these parameters.

Assembly process:

- 1 Same subsystem level check as in any full-vehicle maneuver with possibly an extra check to ensure there are no more than four wheels and two suspensions.
- 2 Assemble vehicle with test rig. Location communicators will locate the pads in the X and Y plane.

## Four-Post Vertical Excitation Test...

- 3 Loop over the tires and reassign the GFORCE reference markers to the appropriate respective pad. The left front tire's reference marker needs to belong to the `gel_front_pad` part of the test rig for example.
- 4 Assign the Z location of the `std_tire_ref` marker based on the average contact patch location of all of the tires (the same as it is done in a full-vehicle simulation).
- 5 Set the tire property file to a hardcoded value of `mdi_2d_flat.rdf` for each of the tires without generating any road geometry.
- 6 Modify the actuator's (`jmf_left_front_actuator`, `jmf_right_front_actuator`, and so on) analytical functions according to the user input data:

The following functions need to be assigned to each actuator based on the analytical drive signal:

Left Front =  $LF\_phase * Peak\_Amplitude * \sin(1/2 * 360D * Freq\_Range / End\_Time * Time^{**2})$

Right Front =  $RF\_phase * Peak\_Amplitude * \sin(1/2 * 360D * Freq\_Range / End\_Time * Time^{**2})$

Left Rear =  $LR\_phase * Peak\_Amplitude * \sin(1/2 * 360D * Freq\_Range / End\_Time * Time^{**2})$

Right Rear =  $RR\_phase * Peak\_Amplitude * \sin(1/2 * 360D * Freq\_Range / End\_Time * Time^{**2})$

Where the following values are assigned to the phase variables:

- Heave Mode: `LF_Phase`, `RF_Phase`, `LR_Phase`, `RR_Phase` = 1.0
- Pitch Mode: `LF_Phase`, `RF_Phase` = 1.0 & `LR_Phase`, `RR_Phase` = -1.0
- Roll Mode: `LF_Phase`, `LR_Phase` = 1.0 & `RF_Phase`, `RR_Phase` = -1.0
- Warp Mode: `LF_Phase`, `RR_Phase` = 1.0 & `RF_Phase`, `LR_Phase` = -1.0

The test rig then goes through the following simulation process:

- Submit the simulation to the solver using a similar process as the full-vehicle simulation. The simulation needs one static equilibrium, an initial velocity = 0.0, and then a dynamic simulation equal to the end. The `hmax` on the integrator should also be set to at least 1/10 of the maximum frequency range. For example, if the frequency range set by the user is 20Hz then the `hmax` should be  $1/10 * 1/20 = 1/200$  (0.005). This is necessary to avoid aliasing of the input during the simulation.

# Four-Post Vertical Excitation Test...

## Creating a test rig template in ADAMS/Car

A minimum prerequisite for the task of adding a test rig to ADAMS/Car is a thorough understanding of all aspects of ADAMS/Car Template Builder. It is very important that users attempting to create test rigs in ADAMS/Car have a firm understanding of the concepts of communicators and actuators. You should reference the guide, *ADAMS/Car Templates* as you work through this section. It might also be beneficial to examine the test rigs that are included in standard ADAMS/Car.

A test rig in ADAMS/Car is almost completely comparable to a template in ADAMS/Car. Test rigs differ from templates mainly because test rigs contain actuator elements, such as motions and forces, to excite the assembly.

The procedure for creating a test rig template in ADAMS/Car is just like the procedure of creating a normal template, with a few differences. The steps for creating a test rig template are:

- Creating a test rig template
- Saving the test rig template
- Modifying the test rig template

## Getting started

A test rig template is created in the template builder in ADAMS/Car. Like a regular template, a test rig template can contain parts attached together via attachments and forces. Unlike most templates, the test rig template will also contain actuators to excite the system. The test rig template, like normal templates, will also contain communicators to enable the exchange of information with other templates.

Because templates and test rigs are so similar, it would be redundant to fully describe how to create test rigs. Instead, see the guide, *ADAMS/Car Templates* for specific information about building templates and communicators.

ADAMS/Car works with test rigs as templates. However, in order to incorporate a test rig for use on an assembly, the test rig must be converted to a test rig model file (.cmd) and a private ADAMS/Car binary created. You can, of course, create a new test rig template from the ADAMS/Car interface very easily, but it is often best to work with existing templates in order to better understand the capabilities of ADAMS.

You start by modifying an existing test rig model file (.cmd) in the template builder. Start by locating the file `acme_3PostRig.cmd`. This is a test rig model file (.cmd) that contains a test rig currently equipped with three active posts.

# Four-Post Vertical Excitation Test...

## Loading a test rig model file (.cmd) into the ADAMS/Car Template Builder

If you want to open an existing test rig model (.cmd) file and edit it using ADAMS/Car Template Builder, follow the steps outlined in this section. Try these steps out on the acme\_3postrig.cmd file. Note that you should be careful if using a private or site ADAMS/Car binary when editing a test rig template. If you already made a test rig and a private binary incorporating it, you cannot edit the test rig of the same name.

### To load a test rig model file:

- 1 Move the test rig model file, **acme\_3postrig.cmd** file, to your private car database under the templates table (for example, C:\private.cdb\templates.tbl).
- 2 Rename the file from **acme\_3postrig.cmd** to **\_\_acme\_4PostRig.tpl**.
- 3 Open the file in a text editor.
- 4 Insert the header as follows:
  - Note that the template name *must* match the file name exactly (excluding the extension).
  - Note that there is a variable called "date" inside the acme\_4postrig.cmd file. This must be set to the same date as the header:

```
$-----MDI_HEADER
[MDI_HEADER]
FILE_TYPE      = 'tpl'
FILE_VERSION   = 13.4
FILE_FORMAT    = 'ASCII'
HEADER_SIZE    = 9
(COMMENTS)
{comment_string}
'ADAMS/Car analysis template'
$-----TEMPLATE_HEADER
[TEMPLATE_HEADER]
TEMPLATE_NAME= '__acme_4PostRig'
MAJOR_ROLE   = 'analysis'
TIMESTAMP    = '1998/08/28,09:23:05'
HEADER_SIZE  = 6
```

## Four-Post Vertical Excitation Test...

- 5 There is a variable inside the `acme_4postrig.cmd` file called `model_class`. Search for `model_class`. Change this variable assignment to look like the following (this variable must be set to template for Template Builder to use it):

```
variable create &  
variable_name = __acme_4PostRig.model_class &  
string_value  = "template" &  
comments     = "Memory for ADAMS/Car model class"  
!
```

### Opening the test rig in ADAMS/Car Template Builder

The file that you just specified as an ADAMS/Car template contains a model of a test rig with only three active posts. In this section, you activate the fourth (left-front) post.

#### To open the test rig:

- 1 Make sure that your `.acar.cfg` file specifies you as an expert user so you can start the ADAMS/Car Template Builder.
- 2 Start ADAMS/Car and select the template-builder mode at the prompt.
- 3 From the **File** menu, select **Open**.
- 4 Right-click the Template Name text box, point to **Search**, and then select `<private>/templates.tbl`.
- 5 Select the `__acme_4PostRig.tpl` template file, and then select **OK** twice.
- 6 Make sure icon visibility is on, the view is set to **Front Iso** and the view is zoomed to fit.



# Four-Post Vertical Excitation Test...

---

## Modifying the test rig in ADAMS/Car Template Builder

To make the actuator active, you will add a variety of items to the test rig: a bumpstop, a reboundstop, a joint attachment, a joint motion actuator, and a joint force actuator.

### To add a bumpstop:

- 1 From the **Build** menu, point to **Bumpstop**, and then select **New**.
- 2 In the **Bumpstop Name** text box, enter `front_extension_stop`.
- 3 Specify the **I Part** as `gel_front_pad` by typing it in or by right-clicking the text box, pointing to **Part**, selecting **Pick**, and then choosing the part `gel_front_pad`.
- 4 Specify the **J Part** as ground by right-clicking the text box, pointing to **Guesses**, and then selecting **ground**.
- 5 Specify the **I Coordinate Reference** as `cfl_Front_Actuator_Base`.
- 6 Specify the **J Coordinate Reference** as `cfl_Front_Pad`.
- 7 Ensure that the **Property File** text box specifies `<shared>\bumpstops.tbl\mdi_0001.bum` as the property file.
- 8 In the **Clearance** text box, enter `127`.
- 9 Select **OK**.

Note that because of symmetry relations, not only is a bumpstop immediately created on the left front actuator base, but one is also created on the right front.

### To create a reboundstop:

- 1 From the **Build** menu, point to **Reboundstop**, and then select **New**.
- 2 In the **Reboundstop Name** text box, enter `front_retract_stop` to enforce a consistent naming convention.
- 3 Set **I Part** to `gel_front_pad`.
- 4 Set **J Part** to **ground**.
- 5 Set **I Coordinate Reference** to `cfl_Front_Actuator_Base`.
- 6 Set **J Coordinate Reference** to `cfl_front_pad`.
- 7 Ensure that the **Property File** text box points to `<shared>\reboundstops.tbl\mdi_0001.reb`.

## Four-Post Vertical Excitation Test...

---

- 8 In the **Clearance** text box, enter 127.
- 9 Select **OK**.

Note that because of symmetry relations, not only is a reboundstop immediately created on the left front actuator base, but one is also created on the right front.

### To add an attachment between the actuator pad and actuator base:

- 1 From the **Build** menu, point to **Attachments**, point to **Joint**, and then select **New**.
- 2 Specify the **Joint Name** as `left_front_pad`.
- 3 Specify the **I Part** as `ground`.
- 4 Specify the **J Part** as `gel_front_pad`.
- 5 Set the **Type** to `single`.
- 6 Set **Joint Type** to `translational`.
- 7 Set **Location Dependency** to `Delta location from coordinate`.
- 8 Set **Location Coordinate Reference** to `cfl_front_pad`.
- 9 Set **Location** to `0,0,0` in `local`.
- 10 Set **Orientation Dependency** to `Delta orientation from coordinate`.
- 11 Set **Construction Frame** to `cfl_front_pad`.
- 12 Set **Orientation** to `0,0,0`.
- 13 Select **OK**.

### To make the joint motion actuator:

- 1 From the **Build** menu, point to **Actuators**, point to **Joint Motion**, and then select **New**.
- 2 Specify the **Actuator Name** as `left_front_actuator` to enforce consistent naming.
- 3 Specify the **joint** as `jostra_left_front_pad` by picking the joint you just created.
- 4 In the **Application** text box, enter `pad_excitation`.
- 5 In the **Identifier** text box, enter `left_front_actuator`.
- 6 In the **Function** text box, enter `15*sin(720d*time)`.
- 7 Specify the **Time Derivative** as `displacement`.

## Four-Post Vertical Excitation Test...

---

- 8 Set the **Force Limits** to -5e4,5e4.
- 9 Set the **Displacement Limits** to -1000,1000.
- 10 Set the **Velocity Limits** to -1e4,1e4.
- 11 Set the **Acceleration Limits** to -9.8e4,9.8e4.
- 12 In the **Units** text box, enter mm.
- 13 Select OK.

### To make the joint force actuator:

- 1 From the **Build** menu, point to **Actuators**, point to **Joint Force**, and then select **New**.
- 2 Specify the **Actuator Name** as If\_force\_actuator.
- 3 Pick the **Joint** as jostra\_left\_front\_pad.
- 4 In the **Application** text box, enter Left front actuator force.
- 5 In the **Identifier** text box, enter Left front actuator force.
- 6 In the **Function** text box, right-click and point to **Function Builder**.
- 7 Paste the following into the text box:

```
(STEP(TIME,0,1,0.002,0)*  
(1e5)*VARVAL(If_actuator_disp))  
+STEP(VARVAL(If_actuator_vel),-pvs_Friction_Saturation_Velocity,  
-pvs_Friction_Saturation_Force,pvs_Friction_Saturation_Velocity  
,pvs_Friction_Saturation_Force)  
-VARVAL(If_actuator_force)
```

- 8 Select OK.

# Four-Post Vertical Excitation Test...

---

## Saving the test rig template

The test rig template is saved to a file just like a regular template. The test rig template should be saved in an ASCII format to facilitate the modifications that are required and described in the next section. Storing the template in ASCII format also ensures portability from one machine to another. It allows, for example, the same file to be used when building a site binary on either a UNIX or NT machines.

### To save the template:

- 1 From the **File** menu, select **Save**.
- 2 Make sure the file format is set to **ASCII**, and that **Zero Adams lds** is selected.
- 3 Select **Close Template**.
- 4 Select **OK**, don't save a backup copy when prompted.

## Making a test rig model file (.cmd) from an ADAMS/Car template

You must manually modify the file of the test rig template to make it into a test rig model file (.cmd). There are two modifications that you must do to the ASCII template file generated from ADAMS/Car.

If you want to take a test rig built in the template builder and then use it as a test rig, you should basically perform the steps in [Loading a test rig model file \(.cmd\) into the ADAMS/Car Template Builder](#) on page 231 in reverse order:

### To make a test rig from a template:

- 1 Outside of ADAMS/Car, from your private database, copy the file **\_\_acme\_4PostRig.tpl** from the **templates.tbl** table to your private directory (C:\acar\_private) or another directory.
- 2 Rename the file to **acme\_4PostRig.cmd**.
- 3 Using a text editor, open **acme\_4PostRig.cmd**.

## Four-Post Vertical Excitation Test...

- 4 There is a variable inside `__acme_4PostRig.tpl` named `model_class`. Change this variable assignment to look like the following (this variable must be set to test rig in order for it to be used as one):

```
variable create &  
  variable_name = __acme_4PostRig.model_class &  
  string_value = "testrig" &  
  comments = "Memory for ADAMS/Car model class"  
!
```

- 5 Remove the header information that is added at the beginning of the ASCII template file. This header must be removed because the command file reader will not understand the information stored in this header and will output errors. A typical header from an ASCII was shown in [Loading a test rig model file \(.cmd\) into the ADAMS/Car Template Builder](#) on page 231.
- 6 Remove all the lines from the beginning of the file up to and including the line containing the `HEADER_SIZE` attribute.
- 7 Save the file

### ADAMS/View variables required in a test rig

Templates and test rigs in ADAMS/Car have information that is stored in ADAMS/View variables to determine how the template is used. All templates, including test rigs have three required variables: major role, minor role, and model class. Test rigs have an additional ADAMS/View variable named assembly class.

All the variables required in a test rig are described below. The first three variables: `role`, `minor_role`, and `model_class` are all created automatically when the test rig template is created. You must manually create the variable `testrig_class`, which is unique to test rigs, as described above.

#### *Major role*

The major role of ADAMS/Car templates and test rigs is stored in an ADAMS/View variable named `role`. The major role of a test rig is always analysis. When creating a test rig in ADAMS/Car, it is important to ensure that this value is set properly.

```
variable create &  
  variable_name = __acme_4PostRig.role &  
  string_value = "analysis" &  
  comments = "Memory for ADAMS/Car major role"
```

# Four-Post Vertical Excitation Test...

## *Minor role*

The minor role of ADAMS/Car templates and test rigs is stored in an ADAMS/View variable named `minor_role`. The minor role of a test rig is typically any. Setting the minor role to any is extremely important if you are designing a test rig that is supposed to work with other subsystems that can have different minor roles. For example, a suspension test rig should work with either front, rear, or trailer type suspensions. If the minor role of the test rig is defined as any in this case, the communicators will not hook up properly between the test rig and suspension subsystem.

```
variable create &  
variable_name = __acme_4PostRig.minor_role &  
string_value = "any" &  
comments = "Memory for ADAMS/Car minor role"
```

## *Model class*

Every model in ADAMS/Car has a specific model class. The model class of a model is stored in an ADAMS/View variable named `model_class`. This variable is automatically created at the time the model is created. Currently, in ADAMS/Car, there are four model classes defined: template, subsystem, test rig, and assembly.

```
variable create &  
variable_name = __acme_4PostRig.model_class &  
string_value = "testrig" &  
comments = "Memory for ADAMS/Car model class"
```

## *Test rig class*

All test rigs in ADAMS/Car can be associated with a particular class of assembly. For example, the `__MDI_SUSPENSION_TESTRIG` test rig is associated with suspension assemblies. The test rig class of a test rig is stored in an ADAMS/View variable called `testrig_class`.

```
variable create &  
variable_name = __acme_4PostRig.testrig_class &  
string_value = "full_vehicle" &  
comments = "Memory for ADAMS/Car testrig class"
```

The `testrig_class` variable is used directly in the graphical user interface. For example, this variable is used in the new suspension assembly and new full-vehicle assembly dialog boxes. These two dialog boxes each contain an option menu from which you select the test rig to be included in the new assembly. This option menu will only contain test rigs that are compatible with that class of assembly. The option menu on the new suspension assembly dialog box, for example, will only list those test rigs that have a test rig class of suspension.

# Four-Post Vertical Excitation Test...

## Preparing to create a private binary

You can now add both your new test rig model file `acme_4PostRig.cmd`) and the macro files attached to this tutorial to build a custom private ADAMS/Car binary which can implement this new test rig. The process of creating private and site binary files is outlined next:

- 1 If the directory does not exist, create a directory `C:\acar_private` (ADAMS always looks for this when running a private binary).
- 2 In this directory, copy the five attached ACSII files into `C:\acar_private`:
  - `acar_build.cmd`
  - `acme_4postrig.cmd`
  - `macros_ana.cmd`
  - `mac_ana_ful_fou_sub.cmd`
  - `acme_four_sim.cmd`

A description of each of these follows:

### **acar\_build.cmd**

This is the file upon which ADAMS will call when building a private binary. In general, this file contains any commands to:

- modify the ADAMS car interface
- import the test rig model files
- add libraries (which show up in the command navigator)
- add macros
- as well as some standard commands which should be in any `acar_build.cmd` file

Reproduced below is the text of the `acar_build.cmd` file:

```
!---- Create custom libraries for storage ----  
library create library_name=.ACME  
library create library_name=.ACME.macros
```

The two lines above create special storage for all the acme macros that will be created.

## Four-Post Vertical Excitation Test...

---

```
!---- Read analysis macros and model ----  
file command read file="C:\acar_private\macros_ana.cmd"  
file command read file="C:\acar_private\acme_4postrig.cmd"  
file command read file="C:\acar_private\acme_four_sim.cmd"
```

It is now necessary for `acar_build.cmd` to call upon any other files that are desired for building a custom binary. In this case, we will call upon three other `cmd` files (described later in this document). Note that these files' location is hard-coded into the `acar_build.cmd` file. The rest of this command file consists of standard commands to undisplay items:

```
!---- Standard command to undisplay model  
model display model=(NONE)  
  
!---- Standard Command for Message Box  
interface dialog execute &  
  dialog_box_name=.gui.msg_box &  
  undisp=yes
```

### **acme\_4PostRig.cmd**

This file contains the test rig model file (`.cmd`) that you just created. This test rig model will be imported into your private binary and now be available to any future assemblies.

### **macros\_ana.cmd**

This file serves as a pointer to `mac_ana_ful_fou_sub.cmd`. It contains a hard-coded file location. It is good practice to use pointers like this rather than to import the simulation macros themselves. This allows for easy modification.

### **mac\_ana\_ful\_fou\_sub.cmd**

This is a macro that instructs ADAMS/Car on how to simulate the model. This file is discussed in depth in [Adding custom analysis procedures to ADAMS/Car](#) on page 247.

### **acme\_four\_sim.cmd**

This is a custom dialog box modification, which also adds a special Acme Four Post Simulation item under the Tools menu. This was built with a text editor, and is commented. The commands that make the menu items appear at the end of the file. It is easy to create your own dialog boxes and menus, but it is often helpful to have a file like this for reference.



# Four-Post Vertical Excitation Test...

---

## Adding the four-post analysis to a private binary

### To create the private binary:

- From the **Start** menu, point to **Run ADAMS 11.0**, point to **ADAMS/Car**, point to **Advanced**, and then enter **cr-privatebin** at the prompt.

## Running ADAMS/Car with a private binary

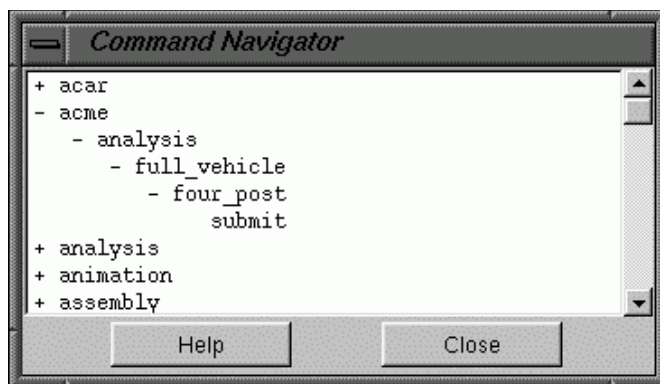
### To run ADAMS/Car with your private binary:

- 1 From the **Start** menu, point to **Run ADAMS 11.0**, point to **ADAMS/Car**, point to **Advanced**, and then enter **ru-private** at the prompt.
- 2 You could also create a shortcut to this containing the path: **C:\Program Files\ADAMS 11.0\common\mdi.bat acar ru-private**.

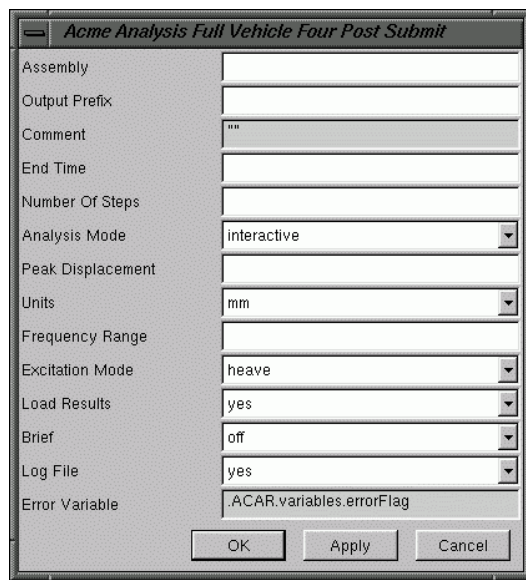
**Note:** When you want to build a binary with a second test rig later, you will have to make sure that **acar\_build.cmd** calls both your new files and these old files!

# Four-Post Vertical Excitation Test...

## Interfaces

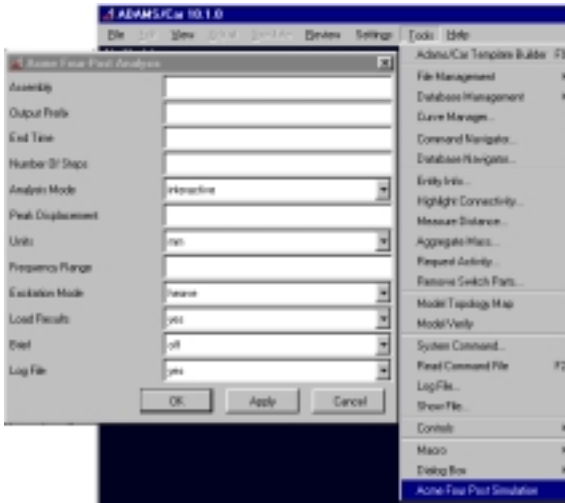


The easiest way to test the four-post analysis macro is to access it from the Command Navigator. The command to be issued is the `user_entered_command` specified in the `acar_build.cmd` file shown above. When you access the four-post macro from the Command Navigator, ADAMS/Car automatically creates a graphical user interface (or dialog box) based on the parameters in the macro. You can use this dialog box to execute the macro and submit the four-post analysis:



# Four-Post Vertical Excitation Test...

It is also easy to create your own interface. Open the file `acme_four_sim.cmd` in a text editor. This is a custom dialog box modification, which also adds a special Acme Four Post Simulation item under the Tools menu. Output is shown next:



## Creating an assembly using the test rig

At this point you should have made a test rig template, test rig model file (.cmd), analysis macro, private binary, and custom interface. You are now ready to run a four-post simulation on a full-vehicle assembly.

### To create an assembly:

- 1 Start ADAMS/Car with your private binary by selecting **ADAMS 11.0 - > ADAMS/Car** -> **Advanced**, from the **Start** menu.
- 2 At the prompt that appears, enter **ru-private**.
- 3 Make sure you select **Standard Interface** if ADAMS/Car prompts you about which interface is desired.
- 4 From the **File** menu, point to **New**, and then select **Full-Vehicle Assembly**.
- 5 In the **Assembly Name** text box, enter **test\_4post\_vehicle**.

## Four-Post Vertical Excitation Test...

---

- 6 Right-click the **Front Susp Subsystem** text box, point to **Search**, point to **<shared>/subsystems.tbl** and left-click to open up a selection window with the various available vehicle subsystems displayed. Select **TR\_Front\_Suspension.sub**.
- 7 Right-click the **Rear Susp Subsystem** text box, and follow the instructions above to select **TR\_Rear\_Suspension.sub**.
- 8 In the **Steering Subsystem** text box, select **TR\_Steering**.
- 9 In the **Front Wheel Subsystem** text box, select **TR\_Front\_Tires.sub**.
- 10 In the **Rear Wheel Subsystem** text box, select **TR\_Rear\_tires.sub**.
- 11 In the **Body Subsystem** text box, select **TR\_Body.sub**.
- 12 Make sure that **Powertrain Subsystem** is not selected.
- 13 Make sure that **Brake Subsystem** is not selected.
- 14 Set **Vehicle Test Rig** to **\_\_acme\_4PostRig**.
- 15 Select **OK**.

It will take some time for the assembly to complete. When it is done, you will still need to switch the tire model to Delft, because it supports zero-velocity friction.

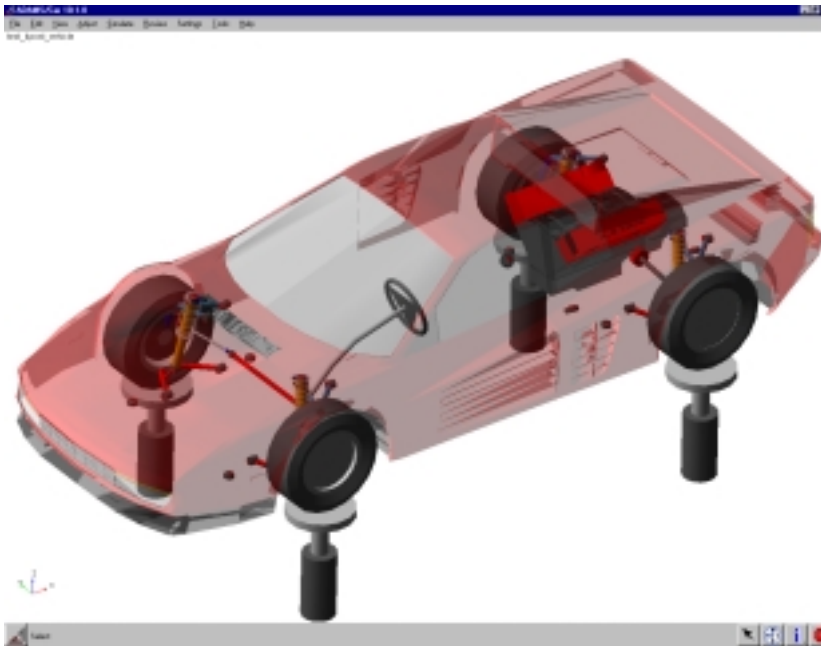
### To switch the tire model to Delft:

- 1 Right-click the mouse on the vehicle's left-front wheel, point to **Wheel:whl\_wheel**, and then select **Modify**.
- 2 Change the **Property File** to **<shared>\tires.tbl\mdi\_delft01.tir**.
- 3 Select **OK**.
- 4 Right-click the mouse on the vehicle's left-rear wheel, point to **Wheel:whl\_wheel**, and then select **Modify**.
- 5 Change the **Property File** to **<shared>\tires.tbl\mdi\_delft01.tir**.
- 6 Select **OK**.

## Four-Post Vertical Excitation Test...

- 7 From the **View** menu, select **Assembly**, and select **OK** to make sure that the top level is selected.

Notice that the wheels on the right side of the car automatically adjust because of symmetry relations. The model should now look something like this:

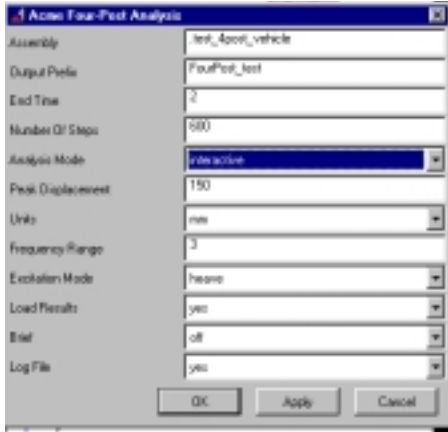


# Four-Post Vertical Excitation Test...

## Running a four-post simulation

### To run a sample four-post simulation:

- 1 From the **Tools** menu, select **Acme Four-Post Analysis**.
- 2 Fill in the dialog box so it looks as follows:



- 3 Select **OK**.

The simulation will take some time to run.

### To review the animation:

- 1 From the **Review** menu, select **Animation Controls**.
- 2 Press the **Play** tool.

You can also review plots of simulation results by selecting **Postprocessing Window** from the **Review** menu. When done, return to **ADAMS/Car** by selecting **Exit**.

You should now save this assembly and exit.

### To save the assembly and exit:

- 1 From the **File** menu, point to **Save**, and then select **Assembly**.
- 2 Select **Close** assembly after save.
- 3 Select **OK**.

**ADAMS/Car** saves this assembly in your private database.

# Four-Post Vertical Excitation Test...

## Modifying existing full-vehicle assemblies

To open the assembly file just created and see how the test rig is specified:

- 1 Outside of ADAMS/Car, find your private database, and open the assemblies table.
- 2 In a text editor, open the file `test_4post_vehicle.asy`.
- 3 Scroll down until you find the text:

```
$-----TESTRIG  
[TESTRIG]  
  USAGE = '__acme_4PostRig'
```

- 4 Close the file.

Note that if you had an existing assembly with the test rig specified as `__MDI_SVT_TESTRIG`, you could just edit the assembly file and change it to read `__acme_4PostRig`.

## Adding custom analysis procedures to ADAMS/Car

A minimum prerequisite of adding an analysis to ADAMS/Car is a thorough understanding of the ADAMS/View macro language. You should also reference *Customizing the ADAMS/Car Interface* on page 11 of the guide, *Customizing ADAMS/Car* as you work through this section.

ADAMS/Car is designed with an open architecture to facilitate extensions based on customer requirements. Most often users desire to extend the analysis capabilities of ADAMS/Car to include an analysis type or maneuver that is typical within their company, but not included in the ADAMS/Car product. All analyses in ADAMS/Car are defined using macros. Therefore, adding a new analysis to ADAMS/Car can be as simple as adding a new macro. A custom graphical interface to the new macro may also be created to facilitate usage, but is not required.

This example will provide you with step-by-step instructions to create your own macro from a given user scenario. The objective of the analysis is described in detail in the attached Four-Post Vertical Excitation Test document. Each of the steps outlined in the above Analysis Macro Structure section will be followed to put together the complete macro. The user scenario is described in an attached Four-Post Vertical Excitation Test case study, and concerns the implementation of a user macro to analyze a full-vehicle mounted on a four-post shaker table with vertical motion inputs.

# Four-Post Vertical Excitation Test...

## Getting started

The best way to get started creating a new analysis is not to start from scratch. There is a broad range of analyses that are possible within the ADAMS/Car product as it is shipped. Examine the existing functionality to find an analysis that closely suites the analysis you want. Once you have selected an existing macro, simply modify it to suite your needs. If you don't know how to examine existing macros in ADAMS/Car, see *Customizing the ADAMS/Car Interface* on page 11 of the guide, *Customizing ADAMS/Car*.

Where possible, existing utility macros should be used in custom analysis macros. Using the existing utilities macros shipped within ADAMS/Car has a few benefits. First, these macros are already written and tested by MDI. Second, using these macros reduces the amount of work required to add a custom analysis to ADAMS/Car. Third, these macros provide a common basis for customization, facilitating maintenance and debugging between different projects.

Any of the standard macros may be viewed via the macro editor inside ADAMS/Car. From the ADAMS/Car menu, choose Tools->Macro->Edit->Modify to bring up the database navigator, at which point you may select the macro you want to display. Selecting `mac_ana_ful_fou_sub` from the database navigator, for instance, will allow you to view the standard macro for submitting a full-vehicle four-post analysis.

## Analysis macro structure

In general, all the analysis macros within ADAMS/Car are structured the same. This is especially true if you are looking only at a particular class of analysis macros. For example, all the open-loop full-vehicle analysis macros are surprisingly similar. The same holds true for all the full-vehicle analysis macros based on the Driving Machine. Typically, a small section within a macro is what makes that macro and the resulting analysis unique. Every analysis macro in ADAMS/Car has the same basic structure.

## Parameter definition

Parameters are the means by which the end user inputs values into the macro. They are placeholders for information that the end user provides when the macro is executed. Macro parameters are described in detail in *Automating Your Work Using Macros* on page 53 of the guide, *Customizing ADAMS/View*.



## Four-Post Vertical Excitation Test...

Two parameters that are common in all analysis macros within ADAMS/Car are the assembly and output\_prefix parameters. The assembly parameter is used to specify which ADAMS/Car assembly within the session will be analyzed. The output\_prefix parameter is used to specify the unique prefix to be added when creating output files specific to the analysis. These parameters are usually defined first in the analysis macro as seen in the example.

The parameters for the four-post maneuver were determined from the attached Four-Post Vertical Excitation Test document. A description is included for the parameters whose values are important for the success of the four-post simulation.

```
! $assembly:t=model
! $output_prefix:t=string
! $comment:t=string:d=""
! $end_time:t=real:gt=0
! $number_of_steps:t=integer:gt=0
! $analysis_mode:t=list(interactive,graphical,background):d=interactive
! $peak_displacement:t=real:gt=0
! $units:t=list(mm):d=mm
! $frequency_range:t=real:gt=0
! $excitation_mode:t=list(heave,pitch,roll,warp):d=heave
! $load_results:t=list(yes,no):u=yes
! $brief:t=list(on,off):u=off
! $log_file:t=list(yes,no):u=yes
! $error_variable:t=variable:d=.ACAR.variables.errorFlag
```

Parameter descriptions:

- **assembly:** This parameter expects the user to specify an existing model.
- **analysis\_name:** A string value indicating the name prefix for all files common to this analysis.
- **end\_time:** A real value telling ADAMS/Solver the end time of the four-post maneuver.
- **number\_of\_steps:** An integer value telling ADAMS/Solver the number of output steps.
- **analysis\_mode:** A string value to specify the mode of the analysis. The two valid modes are interactive or background.
- **peak\_displacement:** The maximum amplitude of the shaker pad vertical displacement.

# Four-Post Vertical Excitation Test...

- units: Hardcoded to mm for tutorial, can be expanded to include other units.
- frequency\_range: A real value indicating the frequency range of the actuator motion functions.
- excitation\_mode: A list value indicating the direction of the shaker pad motions.
- log\_file: Indicates if an analysis log file should be generated.

## Error handling

The error handling section of each analysis macro should contain checks to identify inappropriate assemblies or invalid user-entered values. The error handling section should also contain checks to ensure the assembly contains subsystems that are required to perform a desired maneuver. For example, if you are creating an analysis macro to perform a full-vehicle, straight-line braking maneuver, a check should be performed to ensure that a brake subsystem exists within the assembly. The error handling section should also have checks to ensure that the values specified by the user for specific parameters are realistic.

The four-post analysis must be performed with the `__acme_4PostRig` test rig described in the attached case study. The setup of the assembly and test rig, described in a later section, perform actions based on the elements known to exist in the `__acme_4PostRig` test rig. In addition to verifying that the correct test rig is used, the macro also determines if the analysis name is unique for this assembly.

```
variable set variable_name=$error_variable integer=0

!---- Check to ensure the assembly has the proper testrig ----
if condition=($assembly.original_testrig_name != "__acme_4PostRig")
    acar toolkit warning &
    w="Analysis cannot be submitted!", &
    "The assembly does not have the proper testrig. This analysis only", &
    "works with assemblies using the '__acme_4PostRig' testrig."
    variable set variable_name=$error_variable integer=1
    return
end

!---- Check if analysis name already exists ----
if condition=(db_exists("$assembly.$'output_prefix'_fourpost"))
    if condition=(alert("question","An analysis called
\'$'output_prefix'_fourpost\' already exists. Overwrite it?","Yes","No","",2) == 2)
        variable set variable_name=$error_variable integer=1
        return
    end
end
```

# Four-Post Vertical Excitation Test...

The acar toolkit warning utility macro used in the above example will display the user message in a small window that will remain open until you select OK. This utility macro is used throughout ADAMS/Car to inform the user they have entered an error condition.

## Reading of property files

Many force elements within ADAMS/Car get their nonlinear characteristics from property files. This information must be updated immediately before an analysis is performed. If the property files are not read, the force elements will typically contain inappropriate information that will directly affect the accuracy of the analysis results.

After validation of the assembly, property file information must be read in and assigned. The following example can be used directly within a user's analysis macro. The \$assembly parameter must be an ADAMS/Car assembly as described above. Property files are read in via a user function as described in the following example. This example also demonstrates how to use the acar toolkit message utility macro to descriptive text to the message window. It is very important that the property files are read before the analysis is submitted to ADAMS/Solver.

```
!---- Clear out message window ----
acar toolkit message &
message=" " append=no display=no closeable=yes echo_to_logfile=no

!---- Read property files ----
acar toolkit message &
message="Reading in property files..."
variable set variable_name=$_self.readEm &

integer_value=(eval(read_property_file($assembly)))
acar toolkit message &
message="Reading of property files completed."
```

## Setup of assembly and test rig

The setup of the assembly and test rig is the section of the analysis macro that is unique from other analysis macros. Within this section of the macro, the macro modifies elements of the test rig and assembly based on the type of maneuver being performed and the parameters specified by the user.

The setup of the assembly and test rig is the section of the four-post analysis macro that is unique from other analysis macros. Within this section of this macro, elements of the test rig and assembly are modified specific to the four-post maneuver and user input. The code fragments for the four-post setup are shown below, with a description for each section where needed.

## Four-Post Vertical Excitation Test...

```
!---- Setup the assembly for the maneuver ----
acar toolkit message &
message="Setting up vehicle assembly for four post shaker..."
```

The tire reference markers need to be assigned to the appropriate test pad on the shaker table. The naming conventions of the communicator variables for the reference markers are considered fixed, in that the macro looks for the communicators known to exist in the four-post test rig. Note that the setup of the tire reference markers will only occur once for a particular assembly; if the same assembly is used for multiple four-post analyses, the initial setup will be valid for each analysis.

For each wheel, the tire reference marker is assigned to a shaker pad. The first step is to find each tire in the full-vehicle assembly. The reassignment occurs via an output communicator in the test rig. For more information on communicators, see *Setting Up Communication Between Subsystems and Test Rigs* on page 29, in the guide, *Building Templates*. The communicator holds the name of the part on the shaker pad where the tire reference marker should be attached.

```
if condition=(!db_exists("$assembly.fourpostSetup"))
!---- Parameterize the 4post pad height to the global road height marker just previously
adjusted ----
marker modify &
marker_name=$assembly.testrig.ground.std_tire_ref &
location=($assembly.ground.std_tire_ref.location) &
relative_to=$assembly.testrig.ground
variable set variable=$_self.frontWheel &
object_value=(eval(subsystem_lookup($assembly,"wheel","front")))
variable set variable=$_self.leftFrontWheel &
object_value=(eval(db_filter_name(db_children($_self.frontWheel[1],"ac_tire"),"til_*"))
variable set variable=$_self.rightFrontWheel &
object_value=(eval(db_filter_name(db_children($_self.frontWheel[1],"ac_tire"),"tir_*"))
variable set variable=$_self.rearWheel &
object_value=(eval(subsystem_lookup($assembly,"wheel","rear")))
variable set variable=$_self.leftRearWheel &
object_value=(eval(db_filter_name(db_children($_self.rearWheel[1],"ac_tire"),"til_*"))
variable set variable=$_self.rightRearWheel &
object_value=(eval(db_filter_name(db_children($_self.rearWheel[1],"ac_tire"),"tir_*"))
marker modify &
marker_name=(eval($_self.leftFrontWheel.object_value.ref_marker.object_value)) &
new_marker_name=(eval($assembly.testrig.col_front_pad_mount[1]//".//
$_self.leftFrontWheel.object_value.ref_marker.object_value.name))
```

## Four-Post Vertical Excitation Test...

```
marker modify &
marker_name=(eval($_self.rightFrontWheel.object_value.ref_marker.object_value)) &
new_marker_name=(eval($assembly.testrig.cor_front_pad_mount[1]/"."."//
$_self.rightFrontWheel.object_value.ref_marker.object_value.name))
marker modify &
marker_name=(eval($_self.leftRearWheel.object_value.ref_marker.object_value)) &
new_marker_name=(eval($assembly.testrig.col_rear_pad_mount[1]/"."."//
$_self.leftRearWheel.object_value.ref_marker.object_value.name))
marker modify &
marker_name=(eval($_self.rightRearWheel.object_value.ref_marker.object_value)) &
new_marker_name=(eval($assembly.testrig.cor_rear_pad_mount[1]/"."."//
$_self.rightRearWheel.object_value.ref_marker.object_value.name))
variable set variable=$assembly.fourpostSetup &
integer_value=1
end
```

The motion actuators driving the displacement of the shaker pads must be reset for each individual four-post analysis. This is in contrast to the tire reference marker setup described above, which needs to occur only once for a particular assembly, and remains valid for all successive four-post analyses. The four excitation modes are heave, pitch, roll, and warp. Each of the four shaker pads will have the same magnitude of motion, but a specific excitation mode will determine the direction of the motion. In heave mode, all four shaker pads will move in the same direction. In pitch mode, the front and rear tires will move in opposite directions. For roll mode, the left and right tires have motion in opposite directions. When warp mode is specified, the left front and right rear tires will move opposite to the direction traveled by the right front and left rear tires. The different excitation modes are achieved by specifying a "1" or "-1" multiplier at the beginning of the actuator function definition.

```
!---- Assign actuator functions based on excitation mode ----
!-Heave Excitation
if condition=("$excitation_mode" == "heave")
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_left_front_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_front_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_left_rear_actuator &
```

## Four-Post Vertical Excitation Test...

```
function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_rear_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
!-- Pitch Excitation
    elseif condition=("$excitation_mode" == "pitch")
        acar template_builder actuator set function &
        actuator=$assembly.testrig.jms_left_front_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_front_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_left_rear_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_rear_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
!-- Roll Excitation
    elseif condition=("$excitation_mode" == "roll")
        acar template_builder actuator set function &
        actuator=$assembly.testrig.jms_left_front_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_front_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_left_rear_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
    acar template_builder actuator set function &
    actuator=$assembly.testrig.jms_right_rear_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
!-- Warp Excitation
    elseif condition=("$excitation_mode" == "warp")
        acar template_builder actuator set function &
        actuator=$assembly.testrig.jms_left_front_actuator &
```

## Four-Post Vertical Excitation Test...

```
function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
acar template_builder actuator set function &
actuator=$assembly.testrig.jms_right_front_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
acar template_builder actuator set function &
actuator=$assembly.testrig.jms_left_rear_actuator &

function="-1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
acar template_builder actuator set function &
actuator=$assembly.testrig.jms_right_rear_actuator &

function="1*$peak_displacement*sin(.5*360d*$frequency_range/$end_time*time**2)"
end
```

### Submitting the analysis

A common requirement of submitting the analysis to ADAMS/Solver is the existence of an ADAMS dataset file (.adm) and an ADAMS command file (.acf). The two basic types of analyses within ADAMS/Car are suspension and full-vehicle analyses. In both of these cases, a utility macro is used to generate the ADM and ACF files and submit the analysis to ADAMS/Solver. These utility macros are executed from within each of the suspension or full-vehicle analysis macros.

After setting up the assembly, it is ready to be submitted for the four-post analysis. Since this is a full-vehicle assembly, the corresponding full-vehicle submission utility macro is used. In this case, two additional parameters are specified to have non-default values for the four-post simulation: `generate_road_geometry` and `simulation_type`.

The code fragment for calling the four-post full-vehicle submission macro is shown, with the important associated parameters described below.

```
!---- Perform the analysis ----
acme analysis full_vehicle four_post submit &
assembly=$assembly &
analysis_name="$'output_prefix'_fourpost" &
end_time=$end_time &
number_of_steps=$number_of_steps &
analysis_mode=$analysis_mode &
load_results=$load_results &
brief=$brief &
road_data_file="BEDPLATE" &
generate_road_geometry=no &
simulation_type=fourpost
```

# Four-Post Vertical Excitation Test...

---

Parameter descriptions:

- **assembly:** This parameter expects the user to specify an existing model.
- **analysis\_name:** A string value indicating the name prefix for all files common to this analysis.
- **end\_time:** A real value telling ADAMS/Solver the end time of the maneuver.
- **number\_of\_steps:** An integer value telling ADAMS/Solver the number of output steps.
- **analysis\_mode:** A string value to specify the mode of the analysis. The two valid modes are interactive or background.
- **load\_results:** A string value that specifies if the results of the analysis should be read in after the analysis is complete. Expected values are yes or no.
- **road\_data\_file:** Hardcoded to BEDPLATE to indicate that the car will not be driven across a road surface. ADAMS/Car will internally interpret and understand this hardcoded value.
- **generate\_road\_geometry:** Set to no to indicate that ADAMS/Car should not generate a geometric representation of the data found in the `road_data_file`.
- **simulation\_type:** Hardcoded to fourpost to indicate that the full-vehicle will be attached to a four-post shaker table. ADAMS/Car will internally interpret and understand this hardcoded value to produce the correct `.adm` and `.acf` files.

## Logging the analysis

Many users consider it very important to generate a log describing the analysis that is performed. Generation of the log file is important because it provides historical data that can be stored along with the results of the analysis. The stored data can be useful and is sometimes required to allow a user to regenerate the results of a particular analysis. These utility macros are executed from within each of the suspension or full-vehicle analysis macros.

Utility macros exist that can be used within your custom analysis macro to generate a log file. A utility macro exists for both suspension and full-vehicle analyses. With the analysis completed, the results may be logged to a file. In addition, final diagnostic messages may be displayed to the message window.



## Four-Post Vertical Excitation Test...

---

```
if condition=($log_file)
  acar analysis full_vehicle log &
  assembly=$assembly &
  analysis_name="'$output_prefix'_fourpost" &
  analysis_type="Four Post Shaker Test" &
  analysis_log_file="'$output_prefix'_fourpost.log" &
  comment=$comment &
  end_time=$end_time &
  number_of_steps=$number_of_steps &
  road_data_file="BEDPLATE" &
  initial_velocity=0.0 &
  velocity_units="m_sec" &
  input_parameters="general input" &
  parameter_values=("$number_of_steps")
end

if condition=("$analysis_mode" != "background")
  acar toolkit message &
  message="Simulation is complete."
end
```

### Finishing up

Finally, it is important to ensure all local variables created in the macro using the `$_self` nomenclature are deleted.

```
variable delete variable_name=(eval(db_children($_self,"variable")))
```



**What's in this appendix:**

- [ADAMS/Car Configuration Files, 260](#)
- [ADAMS/Car Data Files, 260](#)

**Table 3. ADAMS/Car Configuration Files**

The file:	Contains:
.acar.cfg	Information that ADAMS/Car reads during startup to correctly initialize the session. There are private, shared, and site configuration files.
acar.cmd	Commands for starting ADAMS/Car.
acarAS.cmd	Preferences you set. The AS stands for <i>After Startup</i> , meaning that ADAMS/Car reads it after it reads other setup files.
acarBS.cmd	Preferences you set. The BS stands for <i>Before Startup</i> , meaning that ADAMS/Car reads it before it reads other setup files.

**Table 4. ADAMS/Car Data Files**

The file:	Does the following:
Aero_force (.aer)	Contains wind-force mappings.
Assembly (.asy)	Lists the subsystems that make up ADAMS/Car assemblies.
Autoflex input (.afi)	Describes a section+centerline+attachment points (an extruded solid). The executable, afi2mnf.exe, processes the file to create a modal neutral file (MNF) flexible body.
ADAMS/Car database (.cdb)	Directory that serves as the ADAMS/Car database.
Driver control (.dcf)	Contains maneuver descriptions for the Driving Machine.
Driver data (.dcd)	Contains data for the Driving Machine.
Differential (.dif)	Defines the slip speed-torque characteristics of a differential.
Driver loadcase (.dri)	Contains driving signals used in a data-driven, full-vehicle analysis. The driver loadcase specifies inputs to the vehicle.
Loadcase (.lcf)	Contains data used in suspension analyses.

**Table 4. ADAMS/Car Data Files (continued)**

<b>The file:</b>	<b>Does the following:</b>
Model (.mdl)	Obsolete
Plot configuration (.plt)	Defines a suite of plots to be automatically generated after completion of an analysis.
Powertrain (.pwr)	Defines the engine speed-torque relationship at different throttle positions.
Property	Contains force properties for the entities: <ul style="list-style-type: none"><li>■ Bump stops (.bum)</li><li>■ Bushings (.bus)</li><li>■ Dampers (.dpr)</li><li>■ Reboundstop (.reb)</li><li>■ Spring (.spr)</li><li>■ Tire (.tir)</li></ul>
Road Data (.rdf)	Contains data on road.
Subsystem (.sub)	Contains information unique to the specific instance of the template the subsystem file references.
Suspension curves (.scf)	Used in the Conceptual Suspension Modeling add-on module.
Steering_assist (.ste)	Contains torsion bar data relating torsion bar deflection to both torque and pressure.
Tables (.tbl)	Subdirectory in the ADAMS/Car database called tables. Each subdirectory contains files for specific types of components, such as springs and dampers, or files for performing tests, such as loadcases and wheel envelopes.
Template file (.tpl)	Defines the topology and major role (for example, suspension or steering) of ADAMS/Car models.
Wheel envelope (.wen)	Contains location vector information that represents the wheel center location and orientation in space. Used for wheel envelope analyses.

