Getting Started with the MapleSim Connector for VI-CarRealTime

Getting Started with the MapleSim Connector for VI-CarRealTime

Contents

List of Tables

Introduction

The MapleSim™ Connector for VI-CarRealTime™ simulation package provides all of the tools you need to prepare and export your dynamic systems models into VI-CarRealTime ANSI-C source code from a MapleSim model. You can create a model in MapleSim, simplify it in Maple™ by using an extensive range of analytical tools, and then generate the source code that you can incorporate into your toolchain.

Scope of Model Support

MapleSim is a comprehensive modeling tool where it is possible to create models that could go beyond the scope of this Connector. In general, the MapleSim Connector for VI-CarRealTime supports systems of any complexity, including systems of DAEs of any index, in any mix of domains.

System Requirements

For installation instructions and a complete list of system requirements, see the **Install.html** file on the product disc.

1 Getting Started

1.1 VI-CarRealTime ANSI-C Code Generation Steps

This chapter describes how to use the MapleSim™ Connector for VI-CarRealTime™ and, in the *Example: Full Powertrain Model (page 7)* section of this chapter, provides a step by step example on how to generate the C code. The MapleSim Connector for VI-CarRealTime template consists of the following steps for generating C code and is described in *Using the Template (page 1)*:

1. Subsystem selection
2. Inputs/Outputs and parameter management
3. C code generation options
4. Generate plugin solver code
5. View generated C code

CRTConnector package

The CRTConnector package is a collection of procedures for manually generating and compiling VI-grade's ANSI-C code from MapleSim models, based on the model's algebraic equations and Dynamic System objects.

For information about the CRTConnector package, enter `?CRTConnector` at a prompt in a Maple worksheet.

1.2 Opening the VI-CarRealTime Plugin Solver Generation Template

To open the VI-CarRealTime Plugin Solver Generation template

1. With your model open in MapleSim, click **Templates** () in the main toolbar and select the **VI-CarRealTime Plugin Solver Generation** template.
2. In the **Attachment** field, provide a worksheet name.
3. Click **Create Attachment**.

   The **VI-CarRealTime Plugin Solver Generation** template opens in a Maple worksheet. Your MapleSim model is displayed in the **Subsystem Selection** window. The **Main** drop-down list in the toolbar shows all of the subsystems in your model.

1.3 Using the Template

The MapleSim Connector for VI-CarRealTime provides a **VI-CarRealTime Plugin Solver Generation** template in the form of a Maple worksheet for manipulating and exporting MapleSim subsystems. This template contains pre-built embedded components that allow you to generate C code from a MapleSim subsystem.

With this template, you can define inputs and outputs for the system, set the level of code optimization, generate the source code, and choose the format of the resulting C code and library code. You can use any Maple commands to perform task analysis, assign model equations to a variable, group inputs and outputs, and define additional input and output ports for variables.

Note: C code generation now handles all systems modeled in MapleSim, including hybrid systems with defined signal input (RealInput) and signal output (RealOutput) ports.

Example models are available in the **VI-CarRealTime Examples** palette in MapleSim.

Step 1: Subsystem Selection

This part of the template identifies the subsystem modeling components that you want to generate C code for. Since VI-CarRealTime only supports data signals, properties on acausal connectors such as mechanical flanges and electrical pins, must be converted to signals using the appropriate ports.

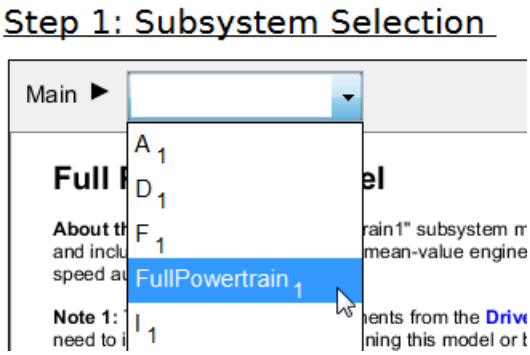
To connect a subsystem to modeling components outside of its boundary, you add subsystem ports to your model. A subsystem port is an extension of a component port in your subsystem. The resulting signals can then be directed as inputs and outputs for the C code files. By creating a subsystem you not only improve the visual layout of a system in model workspace but you also prepare the model for export.

Note: For connectors you must use signal components since acausal connectors cannot be converted to a signal.

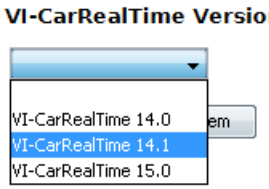
You can select which subsystems from your model you want to create C code for.

To select a subsystem

1. From **Main**, select a subsystem in your model.



2. Select your version of VI-CarRealTime from the **VI-CarRealTime Version** list.



3. Click **Load Selected Subsystem**.

The subsystem appears in the **Subsystem Selection** window. All defined input and output ports are loaded.

Step 2: Inputs/Outputs and Parameter Management

The Port and Parameter Management interface lets you customize, define, and assign parameter values to specific ports. Subsystem components to which you assign the parameter inherit a parameter value defined at the subsystem level.

Select either one of or both of the **Generate external file for assigning parameters** and **Generate external file for assigning initial conditions** options to generate external files for assigning parameters and initial conditions. When selected, a .params file (for assigning external parameters) and an .ics file (for assigning initial conditions) are generated along with your C code. These files can be edited before running your model on VI-CarRealTime to see how different parameters and initial conditions affect your model without having to regenerate the C code for your model.

Inputs:

	Input Variables	Port Grouping Name	Change Row
1	`Main.FullPowertrain1.Wheel_L2_Omega`(t)	"OUTPUT_FV_Wheel_L2_Omega"	
2	`Main.FullPowertrain1.Wheel_R2_Omega`(t)	"OUTPUT_FV_Wheel_R2_Omega"	
3	`Main.FullPowertrain1.driver_demands_throttle`(t)	"OUTPUT_FV_driver_demands_throttle"	

Outputs:

	Output Variables	Port Grouping Name	Change Row
1	`Main.FullPowertrain1.INPUT_FV_mdrv_L1`(t)	"INPUT_FV_mdrv_L1"	
2	`Main.FullPowertrain1.INPUT_FV_mdrv_L2`(t)	"INPUT_FV_mdrv_L2"	
3	`Main.FullPowertrain1.INPUT_FV_mdrv_R1`(t)	"INPUT_FV_mdrv_R1"	
4	`Main.FullPowertrain1.INPUT_FV_mdrv_R2`(t)	"INPUT_FV_mdrv_R2"	
5	`Main.FullPowertrain1.engine_max_trq`(t)	"INPUT_FV_engine_max_trq"	
6	`Main.FullPowertrain1.engine_min_trq`(t)	"INPUT_FV_engine_min_trq"	

☐ Add additional output ports for subsystem state variables

Verify Inputs/Outputs Mapping

Parameters:

Toggle Export Column

	Parameters	Value	Export	Change Row
1	FullPowertrain1_IdlePIDTd	0.5e-1	"X"	
2	FullPowertrain1_IdlePIDTi	2.	"X"	
3	FullPowertrain1_IdlePIDk	2.	"X"	
4	FullPowertrain1_IdleRPMmax	0.9e3	"X"	
5	FullPowertrain1_IdleRPMmin	0.85e3	"X"	
6	FullPowertrain1_Jds	.1	"X"	

☐ Generate external file for assigning parameters

☐ Generate external file for assigning initial conditions

After the subsystem is loaded you can group individual input and output variable elements into a vector array. Input ports can include variable derivatives.

Note: If the parameters are not marked for export they will be numerically substituted.

Step 3: C Code Generation Options

The C code Generation Options settings specify the advanced options for the code generation process.

Solver Options

Select the fixed step solver by specifying the numerical solution method for the model equations during the code generation process.

Select one of the following options:

Euler: *forward Euler* method

RK2: *second-order Runge-Kutta* method

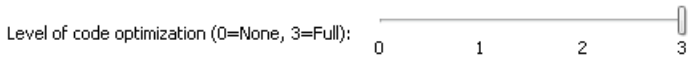
RK3: *third-order Runge-Kutta* method

RK4: *fourth-order Runge-Kutta* method

Implicit Euler: *implicit Euler* method

Optimization Options

Set the level of code optimization to specify whether equations are left in their implicit form or converted to an ordinary differential equation (ODE) system during the code generation process. This option specifies the degree of simplification applied to the model equations during the code generation process and eliminates redundant variables and equations in the system.



Select one of the following options:

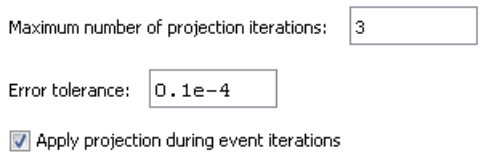
None (0): performs no optimization; the default equations are used in the generated code.

Partial (1, 2): removes redundant equations from the system.

Full (3): performs index reduction to reduce the system to an ODE system or a differential algebraic equation (DAE) system of index 1, and removes redundant equations.

Constraint Handling Options

The **Constraint Handling Options** specify whether the constraints are satisfied in a DAE system by using constraint projection in the generated C code. Use this option to improve the accuracy of a DAE system that has constraints. If the constraint is not satisfied, the system result may deviate from the actual solution and could lead to an increase in error at an exponential rate.



Set the **Maximum number of projection iterations** to specify the maximum number of times that a projection is permitted to iterate to obtain a more accurate solution.

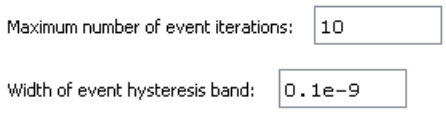
Set the **Error tolerance** to specify the desirable error tolerance to achieve after the projection.

Select **Apply projection during event iterations** to interpolate iterations to obtain a more accurate solution.

Constraint projection is performed using the **constraint projection** routine in the External Model Interface as described on The MathWorks™ web site to control the drift in the result of the DAE system.

Event Handling Options

The **Event Handling Options** specify whether the events are satisfied in a DAE system by using event projection in the generated C code. Use this option to improve the accuracy of a DAE system with events. If the constraint is not satisfied, the system result may deviate from the actual solution and could lead to an increase in error at an exponential rate.



Set the **Maximum number of event iterations** to specify the maximum number of times that a projection is permitted to iterate to obtain a more accurate solution.

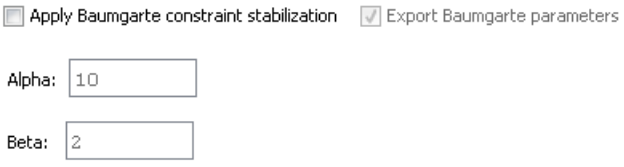
Set the **Width of event hysteresis band** to specify the desirable error tolerance to achieve after the projection.

Event projection is performed using the **event projection** routine in the External Model Interface as described on The MathWorks™ web site to control the drift in the result of the DAE system.

Baumgarte Constraint Stabilization

Select **Apply Baumgarte constraint stabilization** in order to apply Baumgarte constraint stabilization to your model. When selected, you can enter values for the derivative gain (**Alpha**) and the proportional gain (**Beta**) that are appropriate for your model.

Select **Export Baumgarte parameters** to add **Alpha** and **Beta** as parameters in the generated plugin solver code for your model. This allows you to change the values of **Alpha** and **Beta** when using your plugin solver.

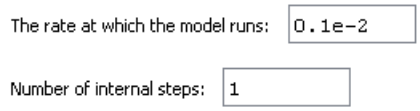


Baserate

The baserate specifies the rate at which the model runs (in seconds). Enter the value for the baserate in **The rate at which the model runs**. Use this option to improve the accuracy of a DAE system with events. If the constraint is not satisfied, the system result may deviate from the actual solution and could lead to an increase in error at an exponential rate. Default is *[0.001]*.

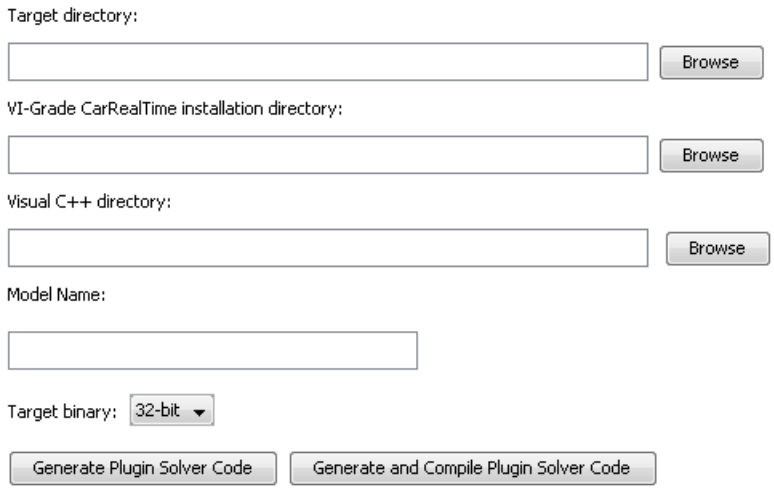
If your baserate is smaller than the step size used in your VI-CarRealTime simulation, you must specify a value in **Number of internal steps** so that:

$$\text{(model baserate)} \cdot \text{(number of internal steps)} = \text{VI-CarRealTime step size}$$



Step 4: Generate Plugin Solver Code

Generating the plugin solver code creates temporary files for viewing purposes in a user defined directory.



To generate plugin solver code

1. Provide the following information for the location and name of the generated code:
   - **Target directory**: Browse to or create the location for the generated C code files.
   - **VI-Grade CarRealTime installation directory**: Browse to the installation directory for VI-CarRealTime.
   - **Visual C++ directory**: Browse to the location of the Visual C++ directory on your computer.
   - **Model Name**: Provide a name for the generated C code folder. This folder is a subdirectory of **Target directory**. Within this folder three files are generated: the VI-CarRealTime interface C code, cMsimModel.h, and a batch file to compile the source code.
2. Select either 32-bit or 64-bit for **Target binary**, depending on the version of VI-CarRealTime you have installed.
3. To generate the plugin solver code, click **Generate Plugin Solver Code**. The C code for the plugin solver is saved in the C code folder.
4. To generate and compile the plugin solver code, click **Generate and Compile Plugin Solver Code**. In addition to the C source code files, object files and a library file (.dll) are created and saved in the C code folder.

Step 5: View Generated C Code

Once the C code is generated, specific portions of the C code can be viewed:

VI-CarRealTime Interface C Code: Displays the code for implementation of the MapleSim Connector for VI-CarRealTime.

MapleSim model: cMsimModel.c: Displays the code for implementation of the MapleSim model.

1.4 Viewing Examples

Within MapleSim there are some examples for you to view.

To view an example

1. Under the **Libraries** tab on the left side of the MapleSim window, expand the **VI-CarRealTime Examples** palette, and then click the entry for the model that you want to view.

Note: Some models include additional documents, such as templates that display model equations or define custom components.

2. Under the **Project** tab, expand the **Attachments** palette and then expand **Documents**. You can open any of these documents by right-clicking its entry in the list and clicking **View**. After you add a template to a model, it will be available from this list.

1.5 Example: Full Powertrain Model

In this example, you will generate C code for a simple powertrain model created in MapleSim.

To generate C code

1. From the **VI-CarRealTime Examples** palette, click the **Full Powertrain** example.
2. Click **Templates** ( ) in the main toolbar. The **Create Attachment for FullPowertrain** window appears.
3. From the template list, select the **VI-CarRealTime Plugin Solver Generation** template.
4. In the **Attachment** field, enter **Full Powertrain** as the worksheet name.
5. Click **Create Attachment**. Your MapleSim model opens in Maple, using the selected template.
6. Select the **FullPowertrain$_1$** subsystem from the **Main** drop-down list in the toolbar above the model diagram.
7. Select your version of VI-CarRealTime from the **VI-CarRealTime Version** list.
8. Click **Load Selected Subsystem**. All of the template fields are populated with information specific to the subsystem displayed in the model diagram. You can now specify which subsystem parameters will be kept as configurable parameters in the generated block.
9. In the **C Code Generation Options > Optimization Options** section, set **Level of code optimization** to **Full (3)**. This option specifies the degree of simplification applied to the model equations during the code generation process, and eliminates redundant variables and equations in the system.
10. In the **Generate Plugin Solver Code** section of the template, specify the **Target directory**, the **VI-Grade CarRealTime installation directory**, the **Visual C++ directory**, and the **Model Name**.
11. Select either 32-Bit or 64-bit from the **Target binary** list.
12. Click **Generate Plugin Solver Code**. The files are created and saved in the C code folder.

Note: Generating a block may require a few minutes.

2 Example: Generating the Plugin Solver Code for the Full Powertrain Model

2.1 Generating the Plugin Solver Code for the Full Powertrain Model

Preparing a Model

In this example, you will perform the steps required to generate the plugin solver code using the Full Powertrain model.

1. Open the Full Powertrain example.
2. Generate the MapleSim Connector for VI-CarRealTime template.
3. Define template settings.
4. Generate the plugin solver code.

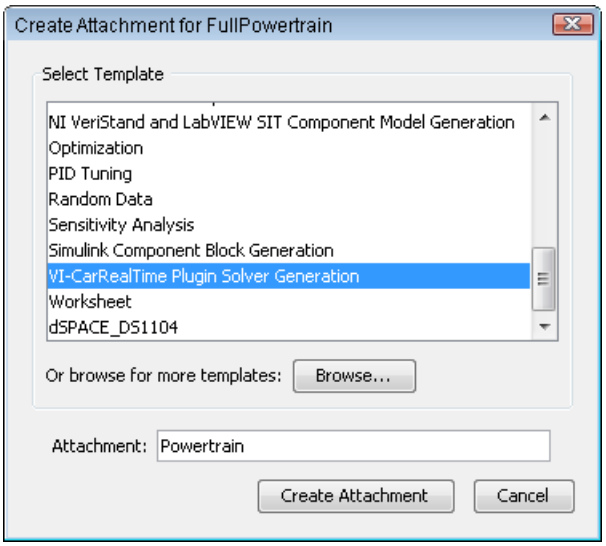
To open the Full Powertrain example

1. In MapleSim, expand the **VI-CarRealTime Examples** palette.

2. Click the **Full Powertrain** example to open it.

To generate the MapleSim Connector for VI-CarRealTime template

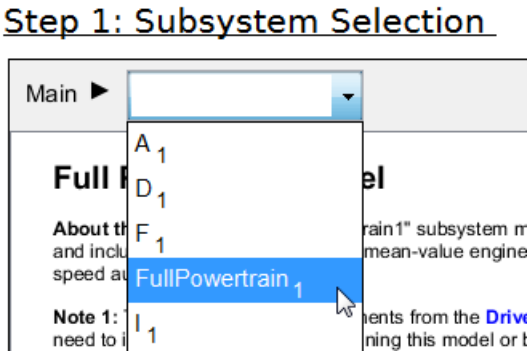
1. If you have not already done so, open the **Full Powertrain** example found in the **VI-CarRealTime Examples** palette.
2. Click **Templates** ( ) in the main toolbar. The **Create Attachment for FullPowertrain** window appears.
3. From the list, select the **VI-CarRealTime Plugin Solver Generation** template.
4. In the **Attachment** field, enter **Powertrain** as the worksheet name.



5. Click **Create Attachment**. Your MapleSim model opens in Maple, using the selected template.

To define the template settings

1. Select the **FullPowertrain$_1$** subsystem from the **Main** drop-down list in the toolbar above the model diagram.



2. Select your version of VI-CarRealTime from the **VI-CarRealTime Version** list.
3. Click **Load Selected Subsystem**. All of the template fields are populated with information specific to the subsystem displayed in the model diagram.
4. In the **Inputs/Outputs and Parameter Management** section, specify which subsystem parameters to keep as configurable parameters in the generated block. The following assignments should be made:
   - **Inputs**: The table below shows the appropriate input variable assignments.

Table 2.1: Input variable assignments for the Full Powertrain model.

Input Variables	Port Grouping Name
`Main.FullPowertrain1.Wheel_L2_Omega`(t)	"OUTPUT_FV_Wheel_L2_Omega"
`Main.FullPowertrain1.Wheel_R2_Omega`(t)	"OUTPUT_FV_Wheel_R2_Omega"
`Main.FullPowertrain1.driver_demands_throttle`(t)	"OUTPUT_FV_driver_demands_throttle"

   - **Outputs**: The table below shows the appropriate output variable assignments.

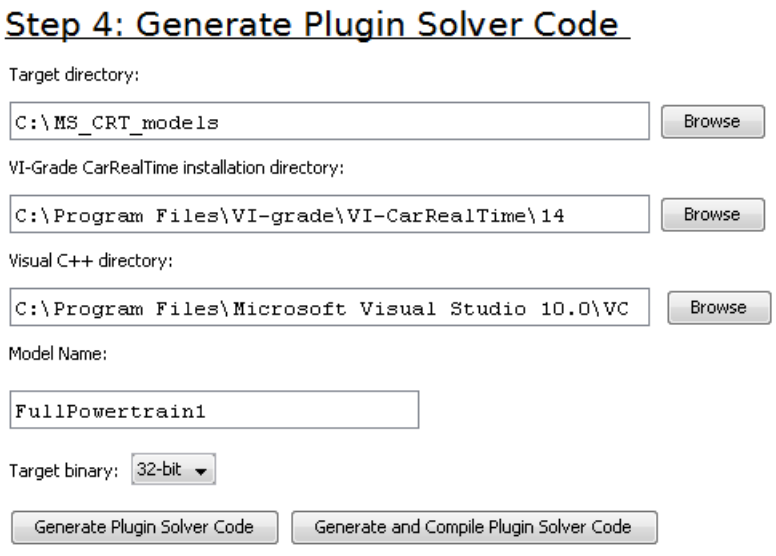
Table 2.2: Output variable assignments for the Full Powertrain model.

Output Variables	Port Grouping Name
`Main.FullPowertrain1.INPUT_FV_mdrv_L1`(t)	"INPUT_FV_mdrv_L1"
`Main.FullPowertrain1.INPUT_FV_mdrv_L2`(t)	"INPUT_FV_mdrv_L2"
`Main.FullPowertrain1.INPUT_FV_mdrv_R1`(t)	"INPUT_FV_mdrv_R1"
`Main.FullPowertrain1.INPUT_FV_mdrv_R2`(t)	"INPUT_FV_mdrv_R2"
`Main.FullPowertrain1.engine_max_trq`(t)	"INPUT_FV_engine_max_trq"
`Main.FullPowertrain1.engine_min_trq`(t)	"INPUT_FV_engine_min_trq"
`Main.FullPowertrain1.engine_omega`(t)	"INPUT_FV_engine_omega"
`Main.FullPowertrain1.engine_trq`(t)	"INPUT_FV_engine_trq"
`Main.FullPowertrain1.transmission_ratio`(t)	"INPUT_FV_transmission_ratio"

5. In the **C Code Generation Options** section, set the following options:

C Code Generation Options	Setting
Solver Options • Fixed step solver	 **Euler**
Optimization Options • Level of code optimization (0=None, 3=Full)	 **3**
Constraint Handling Options • Maximum number of projection iterations • Error tolerance • Apply projection during event iterations	 **3** **0.1e-4** ☑
Event Handling Options • Maximum number of event iterations • Width of event hysteresis band	 **10** **0.1e-9**
Baumgarte Constraint Stabilization • Apply Baumgarte constraint stabilization	 ☐
Baserate • The rate at which the model runs • Number of internal steps	 **0.1e-2** **1**

6. In the **Generate Plugin Solver Code** section of the template, specify the **Target directory**, the **VI-CarRealTime installation directory**, the **Visual C++ directory**, and the **Model Name**.
   The following figure gives an example of some of these settings. Note that the locations of your VI-CarRealTime installation directory and your Visual C++ directory depend on the operating system you are running (XP, Vista, or Windows 7), its version (32- or 64-bit), and the version of VI-CarRealTime.



To generate the plugin solver code

1. Select either 32-bit or 64-bit for **Target binary**, depending on the version of VI-CarRealTime you have installed.
2. Click **Generate Plugin Solver Code** to generate the C code source files. The C source files are created along with a batch file that you can use to compile the source files.
3. Click **Generate and Compile Plugin Solver Code** to generate the C code source files and then compile them.

The generated files are stored in a subdirectory of the **Target directory**. The name of the subdirectory is the same as the **Model Name**. For example, if you entered `C:\MS_CRT_models` for the **Target directory** and `FullPowertrain1` for the **Model Name**, then your C code files are saved in a directory called `C:\MS_CRT_models\FullPowertrain1`.

You can view the generated plugin solver code files (the VI-CarRealTime Interface C Code and cMsimModel.c) in the **View C Code** section of the template.

Note: Generating a block may require a few minutes.

3 Using Your Plugin Solver in VI-CarRealTime

This chapter describes how to import your powertrain model into VI-CarRealTime. You will be using the Full Powertrain plugin solver that you generated in *Generating the Plugin Solver Code for the Full Powertrain Model (page 8)* and the complete vehicle model example that comes with VI-CarRealTime.

Note: For a complete description of VI-CarRealTime, see the VI-Grade **VI-CarRealTime Help**.

3.1 Preparing a MapleSim Model to Run as a New VI-CarRealTime Project

The preparation procedure consists of the following steps:

1. *Loading the complete vehicle model example into VI-CarRealTime (page 12)*
2. *Disabling the example model's powertrain (page 12)*
3. *Enabling your powertrain plugin solver (page 13)*
4. *Configuring the road environment (page 14)*
5. *Running the simulation (page 15)*

3.2 Loading the complete vehicle model example into VI-CarRealTime

The first step in using your powertrain model is to load a vehicle model into VI-CarRealTime. You will be using the complete vehicle model example that comes with VI-CarRealTime.

To load the complete vehicle model example

1. Start **VI-CarRealTime**.
2. If you are not already in Build mode, click the **Build** icon ( ) to enter Build mode.
3. From the **Build** menu, select **Load Model...**.
4. In the **Select File** window, select the **mdids://carrealtime_shared/** database from the **Registered Databases** section.
5. Select the **VI_CRT_Demo_compl.xml** model from the list of files.
6. Click **Open**.

The complete vehicle demo model is now loaded into VI-CarRealTime. This example comes with its own powertrain (that is, engine, clutch, and gearbox systems). Before you can use your powertrain, you have to disable the built-in systems.

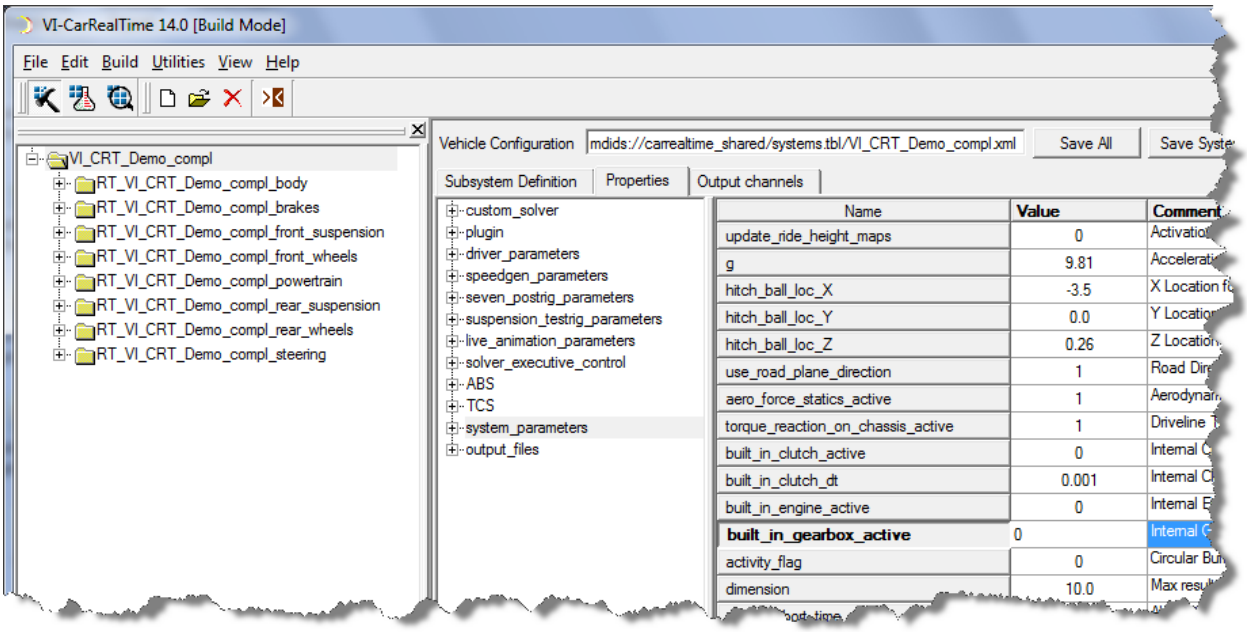
3.3 Disabling the example model's powertrain

To disable the example model's powertrain

1. From the treeview, select **VI_CRT_Demo_compl**.
2. Under the **Properties** tab, select **system_parameters**.
   The **system_parameters** page contains information about your model vehicle setup, components, and general system settings. This is where you will turn off the built-in engine, clutch, and gearbox solvers.
3. Find the names of the following parameters, and set their values to 0 (zero):

Parameter Name	Value
built_in_clutch_active	0
built_in_engine_active	0
built_in_gearbox_active	0

Your **system_parameters** page should look like the following figure.



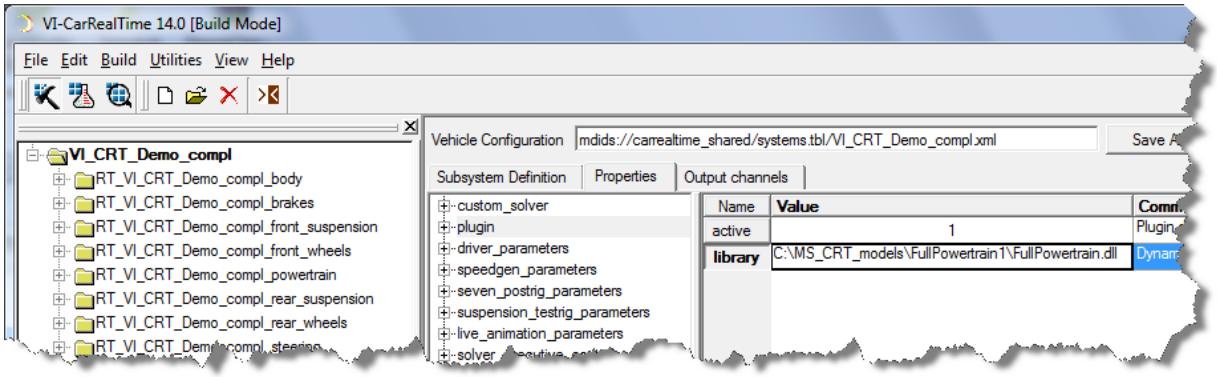
The built-in powertrain is disabled. You are now ready to configure VI-CarRealTime to use the custom powertrain solver that you developed.

3.4 Enabling your powertrain plugin solver

To enable your custom powertrain solver in VI-CarRealTime

1. Under the **Properties** tab, select **plugin**.
2. Set the value of the **active** parameter to 1.
3. Click the Value field for the **library** parameter, and enter the location of the plugin solver library (that is, the dll file) that you generated from your powertrain model.

   The custom solver library is found in the directory that the **VI-CarRealTime Plugin Solver Generation** template stored the generated C code files. For example, if you entered `C:\MS_CRT_models` for the **Target directory** and `FullPowertrain1` for the **Model Name** in the template, then your C code files are in `C:\MS_CRT_models\FullPowertrain1` directory, and your custom library is the file named `FullPowertrain1.dll` in that directory. In this case you would enter `C:\MS_CRT_models\FullPowertrain1\FullPowertrain1.dll` in the **library** field. The following figure illustrates this.



4. Click **Apply**.

Your powertrain model is now configured as the plugin solver for the vehicle model.

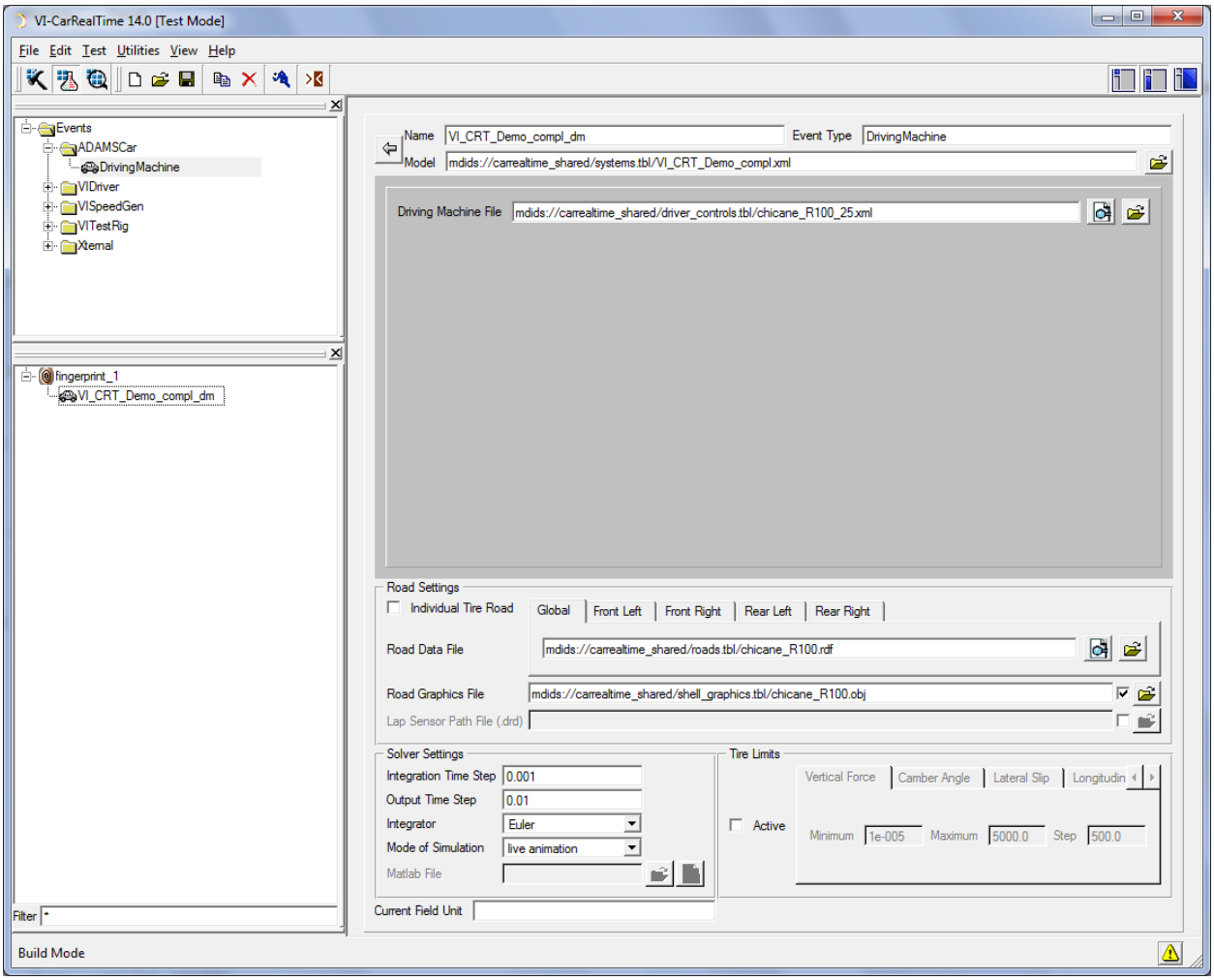
3.5 Configuring the road environment

To configure the road environment for your model

1. Click the **Test** icon ( ) to enter Test Mode.
2. From the **Events** treeview, open **Events**, and then open **ADAMSCar**. From here, select **DrivingMachine**.
3. From the fingerprint treeview, open **fingerprint_1**, and then select **VI_CRT_Demo_compl_dm**.
4. Click the open file icon ( ) next to the **Driving Machine File** input field.
5. In the **Select File** window, select the `chicane_R100_25.xml` file, and then click **Open**.

   Note: If the `chicane_R100_25.xml` file is not available, select the **mdids://carrealtime_shared/** database from the **Registered Databases** section. This also applies for the **Road Data File** and the **Road Graphics File**.
6. Click the open file icon ( ) next to the **Road Data File** input field.
7. In the **Select File** window, select the `chicane_R100.rdf` file, and then click **Open**.
8. Click the open file icon ( ) next to the **Road Graphics File** input field.
9. In the **Select File** window, select the `chicane_R100.obj` file, and then click **Open**.
10. In the **Solver Settings** section, enter the following values:
    - Integration Time Step: **0.001**
    - Output Time Step: **0.01**
    - Integrator: **Euler**
    - Mode of Simulation: **live animation**

The following figure illustrates how your simulation should be configured.

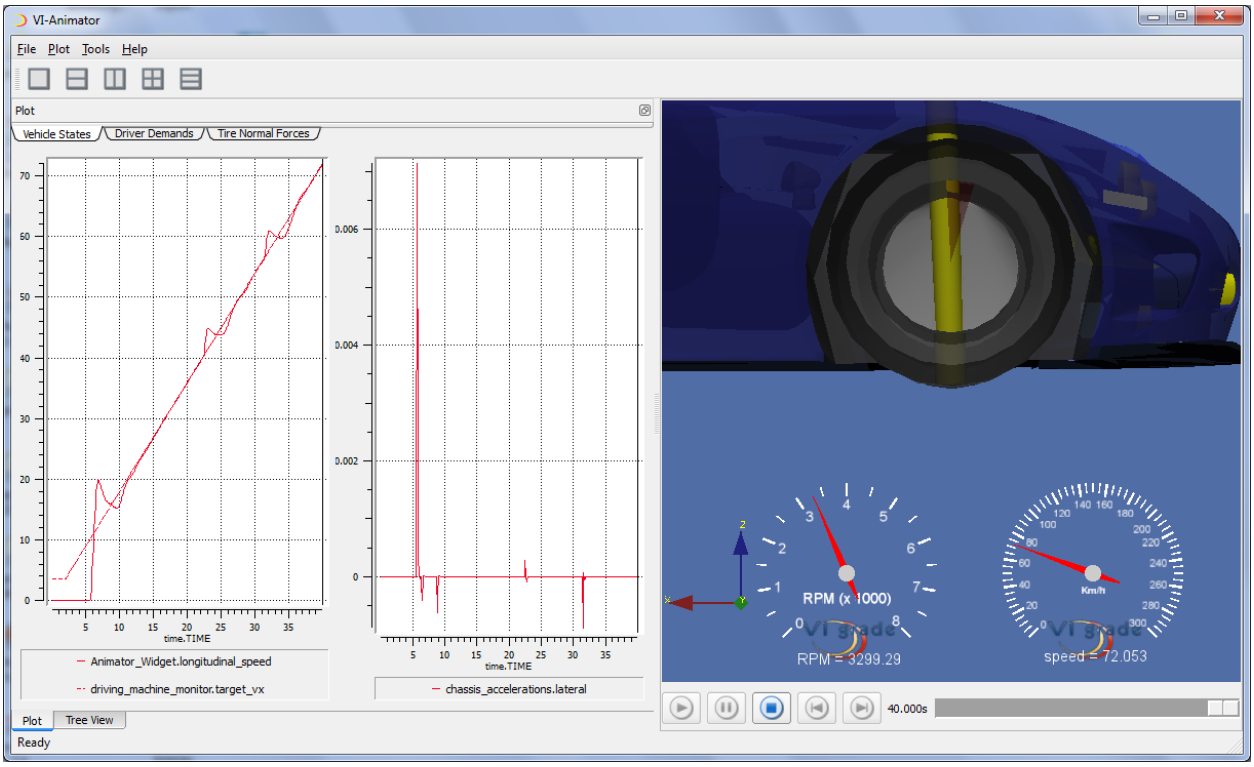


3.6 Running the simulation

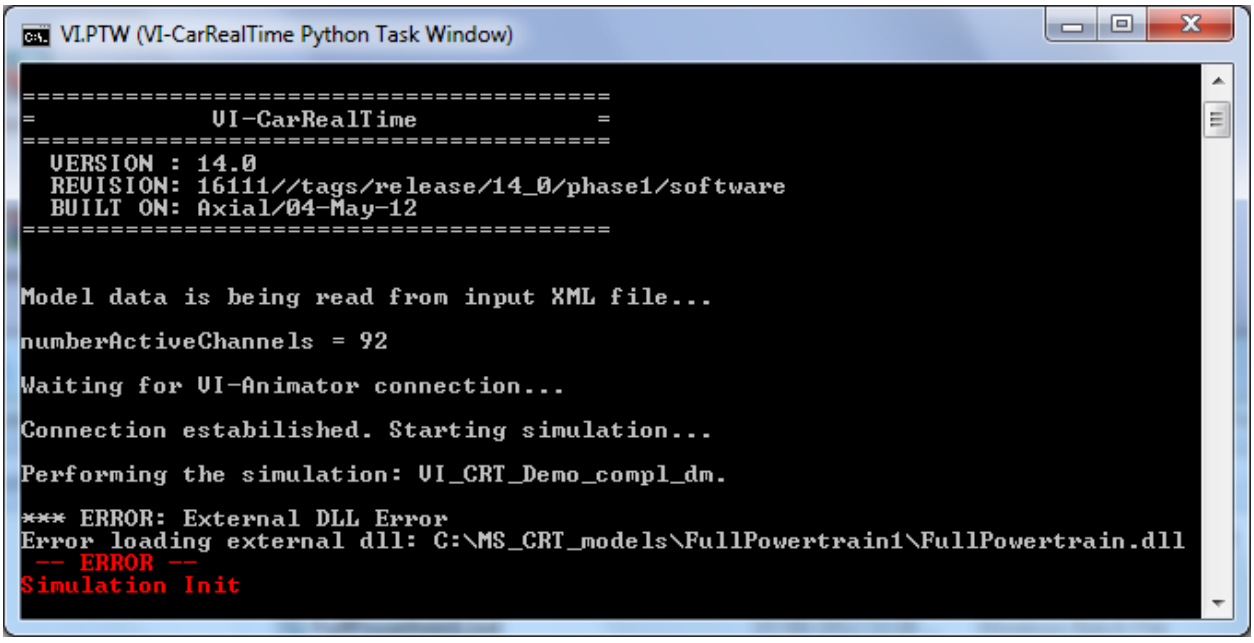
To run the simulation using your custom powertrain

Click the **Run** icon ( ) in the VI-CarRealTime menu bar. Alternatively, from **Test**, select **Run Selected Events**.

The **VI.PTW (VI-CarRealTime Python Task Window)** opens, followed by the **VI-Animator** window (see the following figure). Your simulation starts automatically.



Note: If the simulation does not behave as expected (for example, it does not look as though the simulation is using your custom powertrain), the messages in the **VI.PTW (VI-CarRealTime Python Task Window)** may indicate why the simulation failed. For example, if you did not set the simulation to use your plugin solver properly, you would see a warning message stating that there were problems initializing the user library (see the following figure).



Index