

USER MANUAL

SCANeR™ *studio*

VERSION 1.6

SDK

23 MAY 2017



TABLE OF CONTENTS

I - Modification History of SDK.....	6
II - Contact.....	7
III - SDK Overview.....	8
<i>III.A - Introduction.....</i>	8
<i>III.B - APIs brief description.....</i>	8
<i>III.C - APIs sample list.....</i>	9
IV - SCANeR API.....	16
<i>IV.A - Introduction.....</i>	16
<i>IV.B - Environment.....</i>	18
<i>IV.C - Package.....</i>	20
<i>IV.D - Samples compilation.....</i>	21
<i>IV.E - Aim.....</i>	22
<i>IV.F - Interface of the SCANeR studio API.....</i>	23
<i>IV.G - ANNEXE 1: controller methods.....</i>	37
<i>IV.H - Simulation controller method.....</i>	45
<i>IV.I - ANNEX 2: Lists of data.....</i>	48
<i>IV.J - ANNEXE 3: SAMPLE MODULES IN SCANeR™ studio ENVIRONMENT.....</i>	48
<i>IV.K - Porting from old version.....</i>	79
V - SCANeR API with Simulink.....	82
<i>V.A - Introduction.....</i>	82
<i>V.B - Architecture.....</i>	83
<i>V.C - First steps.....</i>	83
<i>V.D - Model authoring.....</i>	85
<i>V.E - Co-simulation.....</i>	95
<i>V.F - Code generation.....</i>	98
<i>V.G - Samples.....</i>	99
VI - SCANeR API with RTMaps.....	104

VI.A - Introduction.....	104
VI.B - First steps.....	104
VI.C - Diagram authoring.....	106
VI.D - Co-simulation.....	108
VI.E - Limitations.....	110
VII - Analysing tool REPLAY CONTROL API.....	111
VII.A - Introduction.....	111
VII.B - Environment.....	111
VII.C - Package.....	111
VII.D - Samples compilation.....	111
VII.E - Replay control plugin interface.....	112
VII.F - Delivered samples description.....	118
VIII - Scenario API.....	119
VIII.A - Presentation of API.....	119
VIII.B - Use in SCANeR studio.....	124
VIII.C - Scenario special elements.....	127
IX - HTTP API.....	130
IX.A - Introduction.....	130
IX.B - Samples.....	139
X - Vehicle Dynamics Callas API.....	146
X.A - Presentation.....	146
X.B - Definition.....	146
X.C - How it works.....	149
X.D - Vehicle Dynamics Callas API with C++.....	149
X.E - Vehicle Dynamics Callas API with MatLab\Simulink.....	154
X.F - Callas Simulink sub-model (rtw).....	180
X.G - Use an external model on a Vehicle Dynamics Callas.....	183
X.H - Glossary.....	187
X.I - ANNEXE 1: VehicleDynamicsCallas Samples in SCANeR™ studio environment.....	188
XI - Vehicle Dynamics API.....	192

XI.A - Building a vehicle dynamic sample.....	192
XI.B - Building a Simulink vehicle sample.....	197
XII - Vehicle Dynamics ComUDP API.....	205
XII.A - Presentation.....	205
XII.B - Workflow.....	206
XII.C - Reference frames.....	207
XII.D - Vehicle model inputs (driver commands).....	208
XII.E - Vehicle model outputs (Vehicle state).....	209
XII.F - Sample “RemoteVehicleModel”.....	211
XII.G - Run my vehicle model in real-time or in non real-time.....	215
XII.H - How to run an ADAS into my vehicle model?.....	216
XIII - MOTION API.....	218
XIII.A - API Presentation.....	218
XIII.B - Description.....	219
XIV - AFS Interface.....	229
XIV.A - Introduction.....	229
XIV.B - Architecture overview.....	229
XIV.C - Input/Output data definitions.....	231
XIV.D - Developer guide.....	238
XV - VISUAL Plugin API.....	258
XV.A - Introduction.....	258
XV.B - Plugin Development.....	260
XV.C - Installation notes.....	264
XV.D - API functions Reference.....	265
XVI - Image sharing API.....	268
XVI.A - API Presentation.....	268
XVI.B - How does it work.....	268
XVI.C - API Description.....	270
XVI.D - Streaming from Visual and Camera Sensor.....	273
XVI.E - Limitations and constraints.....	277
XVII - UDP RTGateway protocol.....	278

<u>XVII.A - Introduction</u>	278
<u>XVII.B - Protocol</u>	278
<u>XVII.C - SCANeR setup</u>	279
<u>XVII.D - UDPRTGatewayIO sample</u>	280
XVIII - UDP Acquisition protocol	282
<u>XVIII.A - Introduction</u>	282
<u>XVIII.B - Protocol</u>	282
<u>XVIII.C - SCANeR setup</u>	283
<u>XVIII.D - UDPCabin sample</u>	283
<u>XVIII.E - UDP_Cabin_Acquisition sample</u>	284
XIX - VEN API	289
<u>XIX.A - Introduction</u>	289
<u>XIX.B - Environment</u>	290
<u>XIX.C - Package</u>	291
<u>XIX.D - Description</u>	292
XX - VEN Synchronization	302
<u>XX.A - Introduction</u>	302
<u>XX.B - Configuration</u>	303
<u>XX.C - Synchronization protocol</u>	304
<u>XX.D - Messages reference</u>	306
XXI - Indexes	310
<u>XXI.A - Lexical index</u>	310
<u>XXI.B - Illustration index</u>	310
<u>XXI.C - Table index</u>	311
<u>XXI.D - Table of contents</u>	313
XXII - Appendix	326

I - MODIFICATION HISTORY OF SDK

II - CONTACT

If you have any problems, questions or suggestions you can contact the customers support:



OKTAL - Établissement de Paris
19, Bd des Nations Unies
92190 MEUDON
France

Phone:	(33) 1 46 94 93 50
Fax:	(33) 1 41 41 91 44
Technical support team:	support-scaner@oktal.fr
Sales department:	sales@oktal.fr
SCANeR™ web-site:	http://www.oktal.fr/scaner
Bug tracking web-site:	http://support.oktal.fr
OKTAL web-site:	http://www.oktal.fr
RoadXML® web-site:	http://www.road-xml.org
RoadXML® contact:	contact@road-xml.org

III - SDK OVERVIEW

III.A - INTRODUCTION

The SCANeR SDK is a set of different APIs. Each API is dedicated to a specific area of the SCANeR simulation.

The aim of this overview is to briefly present the different APIs, their goals and basic usage and, theirs samples. A longer description of each API is available in each API dedicated section.

III.B - APIs BRIEF DESCRIPTION

API	Description
SCANeR API	Main api, used to create custom SCANeR process. Usually used for acquisition process. Available in C/C++, python, c#/.net, Matlab Simulink, Labview.
AFS API	Pilot the vehicle headlight with scenario environment as input. Available in C/C++, Matlab Simulink.
Motion API	Create plug in to the SCANeR motion process. 2 kind of plug ins are available: <ul style="list-style-type: none">Communication: to exchange data with a motion platformStrategy to compute motion command from the SCANeR vehicle output. Available in C++, Matlab Simulink.
Scenario API	Modify create SCANeR scenario file. Usually used to generate scenario file from a template for automatize simulation. Available in C/C++, python, Matlab Simulink, Labview.
HTTP API	Is to use the HyperText Transfer Protocol to remotely access to some of the supervisor functions and to some of the AnanlysingTool functions
VehicleDynamics API	Create your own vehicle model. Usually used to integrate an external already existing model. If you only need to customize an element of a vehicle look the VehicleDynamicsCallas API. Available in C/C++, Matlab Simulink.

API	Description
VehicleDynamicsCallas API	 Previously this model was sometimes called Advanced or Prosper, starting with SCANeR™ studio 1.4 it is simply called Callas This API enables replacing a part of the Callas vehicle model such as the Engine or the transmission. It can also be used to modify vehicle inputs to simulate an ADAS. Available in C++, Matlab Simulink.
UDP Acquisition	This is not an API, it is a protocol. This protocol defines how to connect a custom made Acquisition module to SCANeR studio HumanDriver. This is an alternative to writing a SCANeR API acquisition. The advantage of this solution is that the standard mapping and calibration tools of the HumanDriver module can be used instead of being implemented in the acquisition. The other advantage of this solution is that the acquisition can be deployed once and work on all SCANeR versions without modification/recompiling.
VEN API	API to the VEN communication protocol. Typical usage is to have a way to communicate easily between non SCANeR process that need to interact with the SCANeR simulation and the SCANeR processes. Available in C/C++, python, Matlab Simulink, Labview.
Visual Plugin API	Add custom effect to the SCANeR visual process. For example add a sphere on top of a target vehicle. Available in C++.
ImageSharing API	Retrieve a stream from the SCANeR visual and camera sensor processes for post processing. Available in C/C++

III.c - APIs SAMPLE LIST

III.c.1 - AFS API

Sample Name	Available	Sample Description
DynamicAfsSamle	C/C++	Modulate the right or left light intensity and heading angle from the steering wheel angle
FullAfsSample	C/C++	Modulate the right or left light heading and pitch angle from the steering wheel angle

Sample Name	Available	Sample Description
StaticAfsSample	C/C++	Modulate the right or left light intensity from the steering wheel angle
Dynamic	Matlab Simulink	The heading angle of the left strategy, and to the pitch angle of the right strategy match the angles of the front wheels.
Static	Matlab Simulink	The intensity of the left strategy match the angles (absolute value) of the front wheels.
Blinding	Matlab Simulink	The light is directed straight at oncoming traffic to blind them. This sample demonstrates using the position of the other vehicles.

III.C.2 - MOTION API

Sample Name	Available	Sample Description
DirectReplay	Matlab Simulink	Simple position control strategy
MotionSampleCom	C++	Accept all orders and give empty return data.
MotionSampleStrategy	C++	Forwards the virtual vehicle movement to the motion platform
PassThrough	Matlab Simulink	Forwards the virtual vehicle movement to the motion platform

III.C.3 - SCANE R API

Sample Name	Available	Sample Description
event_laser_meter	Python	Print all laser meter sensors
ExportChannels	C++ QT	Qt sample, Display process state and send / display export channel.
FlashControl	C++ Flash Qt	Flash/SCANE R API/QT sample. Print export channel in a flash animation.
HighLevel	C#	Send and display export channels using the high level C# binding of the SCANE R API.
initial_state	Python	Print scenario initial conditions.

Sample Name	Available	Sample Description
LaneKeeping	Matlab Simulink	Send vibrations on the steering wheel when the driver diverges from the lane center
LaserMeterViewer	C++ Qt	Qt sample. Display vehicle sensor values.
LowLevel	C#	Auto ignition of the vehicle 0 is the engine state is off. Display vehicle Use the 0 Engine values and export channel number 5000. Use the C# low level binding of the SCANeR API.
Manage_Process	LabView	
Network_Sample	LabView	
ObjectsViewer	C++ Qt	Qt and RoadXml sample. Display the objects define in the RoadXml used by a scenario. Object parameters are interactively editable.
RVLV	Matlab Simulink	
sample_event_path	Python	Print vehicle path with events.
sample_event_physic	Python	Print collision and physic objects position.
sample_path	Python	Print vehicle path without events.
SampleMatLab	MatLab	Displays position of vehicles in a MATLAB plotter.
SampleDynamicHeightMap	C/C++	Module to manage a dynamic_ground object (scenario resource)
SampleMatLabLaserMeter	MatLab	Gets laser meter data and displays ray-tracing detection in a MATLAB plotter.
ScanerAPISampleComClientInterface	C/C++	Display speed, steering angle indicator, gearbox and wheel angle parameter of vehicle 0 and 1. Send export channel 20 and 21.
ScanerAPISampleComModelShm	C/C++	Basic Acquisition sample. Read/write vehicle to cabin and cabin to vehicle input shared memory
scanerAPISampleEvent	C/C++	SCANeR API event handling sample. Handle and validate Kill, Go, Pause Load ans start event. Handle event for vehicle update for only vehicle 0 and 1. Handle event for all vehicle model inputs.

Sample Name	Available	Sample Description
scannerAPISampleMiceAcquisitionMapping	C/C++	Retrieve vehicle output from SHM and send then through export channel to be read by a SCANeR script.
scannerAPISampleProcess	C/C++	Very basic SCANeR API process. Can be used as a base for any other SCANeR API process.
scannerAPISampleRadar	C/C++	Print all radar informations and vehicle update informations using SCANeR API events.
scannerAPISampleSimulation	C++	SCANeR API Launcher sample. This sample demonstrate the use of the SCANeR API to manage a SCANeR simulation (Launch process, launch a scenario, stop a scenario..). It can replace the SCANeR main GUI.
scannerAPISampleSpy	C++	Display one of the following SCANeR message, the choice is ask at run time: VehicleUpdate, ModelInput, ExportChannel, CabToModel
ScannerAPISampleTrafficLight	C++	Manage the state of 4 traffic lights.
scannerAPISampleWeatherControl	C++ MFC	Control all weather parameters available in SCANeR with a GUI.
SHM_Sample	LabView	
SpeedLimiter	Matlab Simulink	Speed limiter for vehicle 0. Read the vehicle input SHM and use corrective SHM to control the vehicle speed.

III.C.4 - SCENARIO API

Sample Name	Available	Sample Description
scenarioAPISampleScannerAPI	C/C++	Modify value contains in a SCANeR .scenario and save the scenario as a new one. Vehicle model, vehicle initial speed, vehicle abscissa on road, and script variable value.
scenarioAPISample	Python	Modify a scenario (vehicle initial speed) and save it.

III.C.5 - HTTP API

Sample Name	Available	Sample Description
http://localhost:9091		use the HyperText Transfer Protocol to remotely access to some of the AnalysingTool functions
http://localhost:9090		use the HyperText Transfer Protocol to remotely

Sample Name	Available	Sample Description
		access to some of the Supervisor functions

III.C.6 - VEHICLE DYNAMICS API

Sample Name	Available	Sample Description
rqSample	C++	Custom road query plug in sample. Always return 0 Z and (1, 0, 0) normal.
SampleDriver	C++	Custom driver plug in sample. Return full acceleration and a steering wheel angle depending of the time.
SampleDriverUI	C++	Driver configuration UI sample. To be used with the previous driver sample. Displayed in the main SCANeR GUI. The UI is empty.
VehicleDynamicsSample	C++	Basic vehicle dynamics model sample.
VehicleEditionSample	C++	Vehicle configuration UI sample. To be used with the previous vehicle model sample. Displayed in the main SCANeR GUI. The UI is just a manual edition of the configuration file.
VehicleDynamicsSimulink	Matlab Simulink	Basic vehicle dynamics model sample.
VehicleDynamicsSimulinkExt	Matlab Simulink	

III.C.7 - VEHICLE DYNAMICS CALLAS API

Sample Name	Available	Sample Description
ExternalDefects	C++ Matlab Simulink	Vehicle module to simulate defects on the SCANeR Callas model.
SimpleEngine	C++ Matlab Simulink	Simple engine to replace the default engine of the SCANeR Callas model.
SinusPilot	C++ Matlab Simulink	Vehicle module to modify the steering command of the vehicle. Return a steering depending on time.
SinusPilot_C	C	C version of the SinusPilot sample

III.C.8 - UPD API

Sample Name	Available	Sample Description
UDPCabin	C/C++	Send UDP packets containing values generated from time dependant functions (ex: sinus). Replace the time dependant functions by your acquisition values to create a UDP acquisition module.
UDPRTGatewayIO	C/C++	Send/Receive UDP packets to/from RT_GATEWAY module.

III.C.9 - VEN API

Sample Name	Available	Sample Description
Basic	C/C++ Python	Read a message of integer and write a message of double precision floating point value.
Interpreter	C++	Read a command file and send VEN message from these commands.
scenario	Python	Write message to 2 readers, write to one.
VEN_Sample	LabView	Send and receive VEN message with Labview

III.C.10 - VISUALPLUGIN API

Sample Name	Available	Sample Description
OsgGLPlugin	C++	Draw a square on quarter of the viewport. Pure OpenGL sample.
OsgPlugin	C++	Attach a spere to a vehicle. OSG recommended sample.
OsgInfrastructurePlugin	C++	Retrieve all infrastructure objects from the Visual.

III.C.11 - IMAGE SHARING API

Sample Name	Available	Sample Description
ImageSharingSample	C++	Gets the frames rendered by the Visual Module and a CameraSensor module to re-display them in a new window. Uses OpenCV

IV - SCANER API

IV.A - INTRODUCTION

SCANeR™ studio is an open software platform that provides to customers the ability:

- to develop their own modules which will communicate with the rest of the simulation system without having to modify the core of the code source or
- to connect their personal software or hardware taking into account the general framework.

The dynamic library SCANeR™ studio API is delivered in order to open the software to the user applications. This new API is a grouping of all APIs available with SCANeR™ studio ; with a common way to use it. One and only one global SCANeR_API is created in order to simplify all the developments made using the SCANeR™ studio's API.

The figure below shows the architecture of the SCANeR_API:

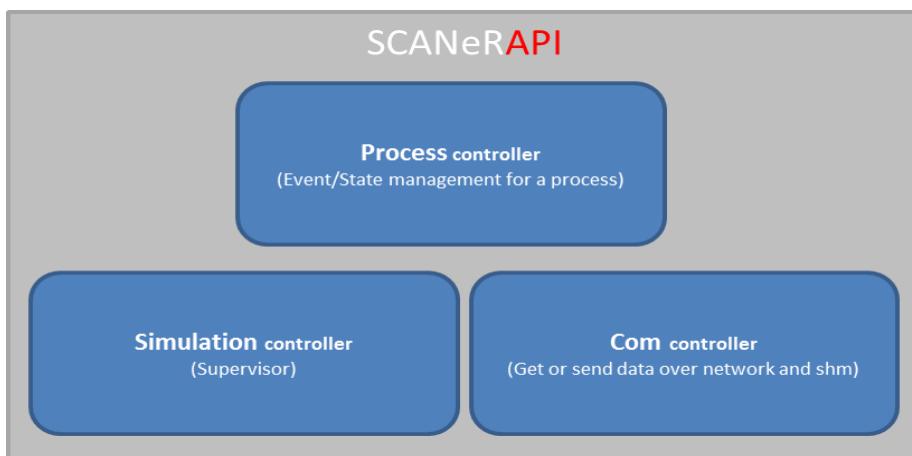


Illustration 1 : The SCANeR™ studio API architecture

To be compliant with a lot of application, the new SCANeR_API is delivered as a DLL (which contains all needed methods), with a “C language” binding. It can be mapped with applications written in C, C++, C# or any application able to use a DLL.



It means that you need to start the demon with the same configuration as your own module was built with.

In an easy way and without high programming background requirements, the following actions can be done:

- connecting SCANeR™ studio to external applications (LabView, Rapid++, MatLab/Simulink ...), including devices in the simulation loop
- developing SCANeR™ studio modules in C or C++
- creating personalized launchers to manage SCANeR™ studio modules
- creating an acquisition module, or a data display module, control autonomous vehicles ...

or more basically, send and receive numerical values through channels.



SCANeR API is not thread safe.

IV.A.1 - ADVICE

When you find problem because you don't know how to set arguments, we advise to:

1. create a basic scenario with a MICE script which replicates the behaviour
2. launch the **CONTROLPAD** in order to spy the Network messages between modules (dedicated tab)
3. analyse the arguments of the Network message. This will help you to know the arguments to use.

IV.B - ENVIRONMENT

IV.B.1 - OPERATING SYSTEM

Microsoft Windows Vista/7/8

IV.B.2 - COMPILER

Microsoft Visual Studio C++ 2013 Update 3

IV.B.3 - THIRD PARTY LIBRARIES

IV.B.3.i - Qt

For GUI: Qt 5.3.2 (Compiled with Compiler described: chapter IV.B.2 - page 18)

IV.B.3.ii - OSG

For SCANE API visual: OSG 3.0.1 (Compiled with Compiler described: chapter IV.B.2 - page 18)

IV.B.3.iii - Matlab/Simulink

SCANE SDK is compatible with Matlab R2011a and R2013b.

To build a Simulink model a compatible compiler with Simulink R2013b product must be used (*you can refer to the following website: <http://fr.mathworks.com/support/compilers/R2013b>.*)



When you use SfonctionCallasMem.dll and SfonctionCallasNet.dll make sure to use the same architecture as your MatLab (32 or 64 bits).



Users should consider the platform (32/64bit) before the specification, before the computer purchase and before the Microsoft Office purchase.

IV.B.3.iv - OpenCV

For ImageSharing API version 2.4.8 (Compiled with Compiler described: chapter IV.B.2 - page 18)

IV.B.3.v - RTMaps

SCANE SDK is compatible with RTMaps 4.1 (32 bit) and 4.2 (32 and 64 bit).

More details about RTMaps packages are available in VI - "SCANE API with RTMaps" - page 104. The RTMaps packages are not provided the SCANETMstudio installer, if you are interested in RTMaps

usage please contact the Oktal support team at support-scaner@oktal.fr to request the RTMaps packages.

IV.B.3.vi - Python

For the ScanerAPI/python samples you need a Python 2.7 (or upper version)

IV.C - PACKAGE

The SCANeR™ studio API package includes:

- A DLL file: **SCANeR_API_C.1.X.dll** (located in **<STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013**) & .
- A LIB file: **SCANeR_API_C.lib** (located in **<STUDIO_PATH>\<STUDIO_VERS>\APIs\lib\<PLATFORM>\vs2013**)
- An include file, **scannerAPI_DLL_C.h** (located in **<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\ScannerAPI**)
- This documentation
- A MS Visual C++ 2013 solution which contains some examples written in C and C++

IV.D - SAMPLES COMPILATION

Get the needed third party software according to your needs (find more information through chapter “I.A Environment”) or ask to OKTAL the corresponding package.

In the <STUDIO_PATH>\<STUDIO_VERS>\APIs\samples folder edit (with text editor) the APIs.props file to update paths according to your environment (location of third parties software).

Open “complete.sln” with Visual Studio 2013 and compile needed samples. Execution of compiled samples may need additional DLL from third party software so do not forget to copy the needed DLLs in the folder containing your executable file or add the DLL folder to your PATH environment variable.

IV.E - AIM

The aim of this API is to allow users to customize SCANeR™ studio for their own applications: creating personalized launchers to manage SCANeR™ studio modules, creating specific modules, connecting SCANeR™ studio to external applications (LabView, Rapid++ ...), including devices in the simulation loop ...

With this API, users can easily, and without much programming background, do data exchange with SCANeR™ studio simulation. More specifically, the SCANeR API provides the possibility to:

- Import / Export Channels,
- Access to autonomous vehicles data (position, velocity,...),
- Access to interactive vehicles data (position, velocity, input, output,...),
- Access to the acquisition shared memory,
- Access to Eye Tracker data,
- Open SCANeR™ studio Core to external software or hardware

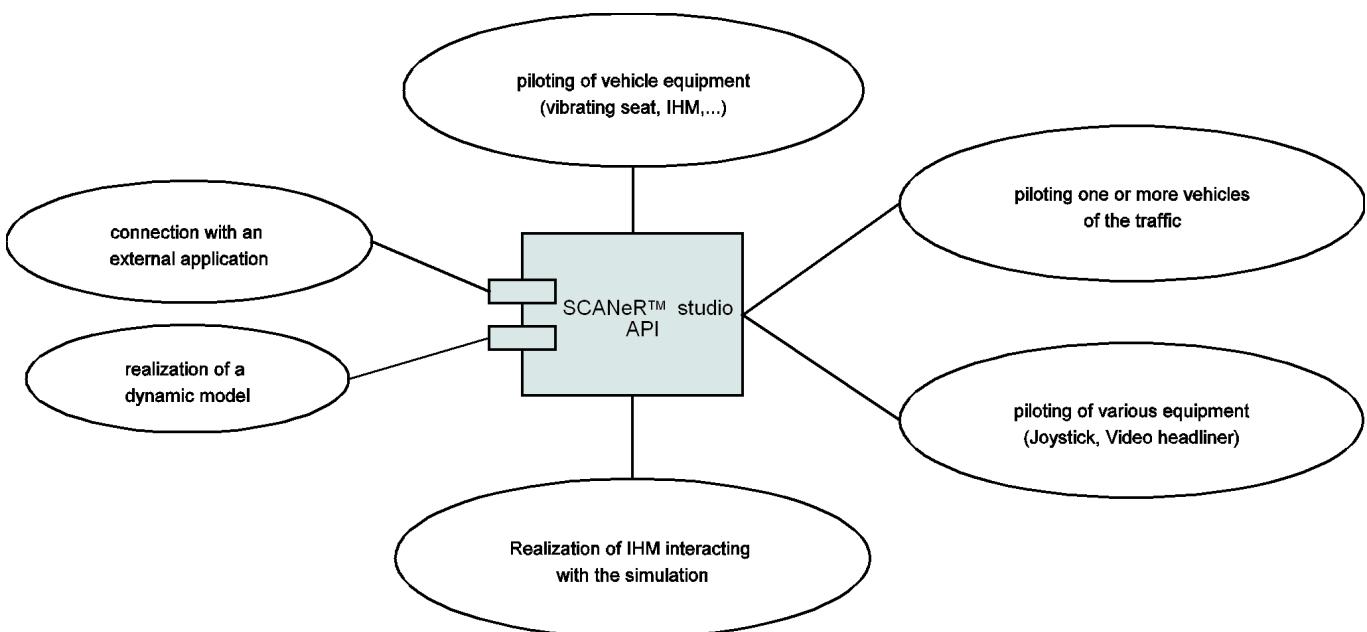


Illustration 2 : Use Case of the SCANeR™ studio API

IV.F - INTERFACE OF THE SCANeR STUDIO API

The SCANeR™ studio API is composed of three functionality groups:

- The ‘Process’ controller
- The ‘Simulation’ controller
- The ‘Communication’ controller

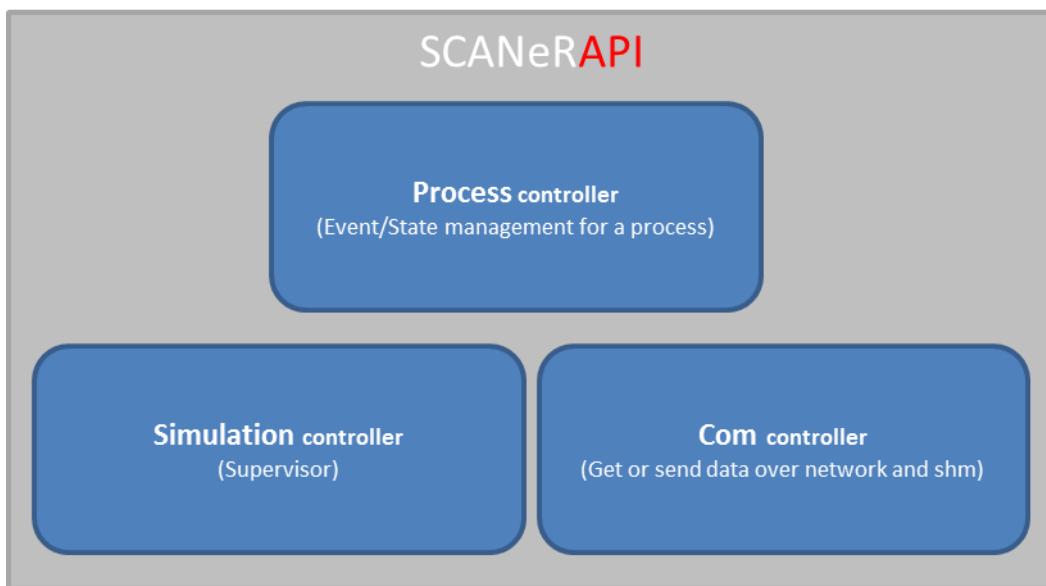


Illustration 3 : SCANeR studio API

IV.F.1 - WHAT IS A SCANeR STUDIO MODULE

IV.F.1.i - State management of the new module

A new module built with SCANeR™ studio API has the same state management as existing modules. Once it is added to the *Configuration* by the SCANeR™ studio *Configuration Manager*¹, it is used like any existing module with SCANeR™ studio.

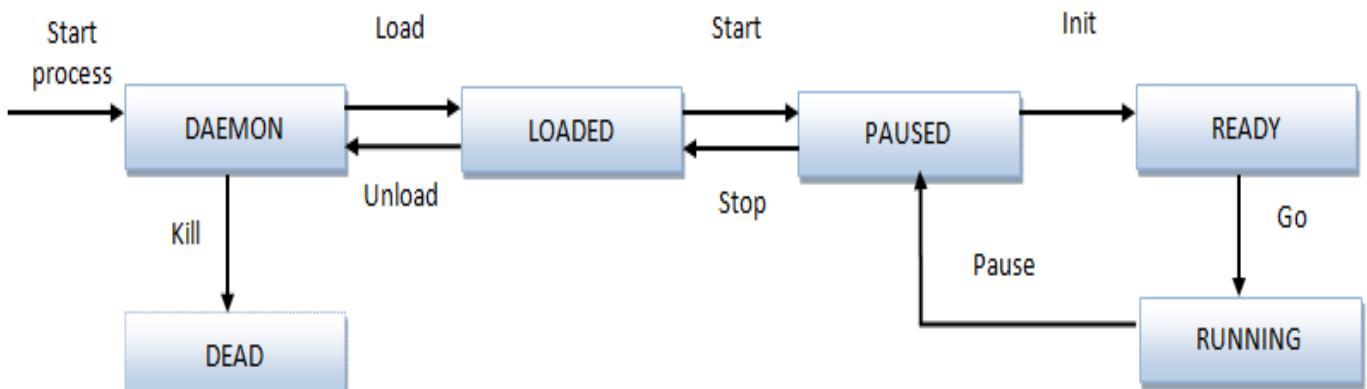


Illustration 4 : SCANeR™ studio state management

The state is changed automatically by the API, when an external control message is received (from the SCANeR™ studio Simulation mode for example). The current state can be given by the API using the `Process_GetState()` method.

Using the [Communication controller](#), the user will be able to be informed of the module state changes (by registering the chosen states), and will then be able to do all the relevant actions he has to perform between these state changes (for instance initialization of some components when a simulation is loaded).

IV.F.1.ii - How a SCANeR Module is started

To be able to be started, an official SCANeR™ studio module must know the name of the current configuration and its process name in this configuration to be able to get access to the associated configuration files or data.

This informations are given to the executable of the module by launching him with some specific arguments (automatically done when a module is launched from the SCANeR™ studio SIMULATION Interface) :

`SCANeRSTUDIOModule.exe -c NameOfTheConfiguration -p NameOfTheProcess`

An additional argument (- f Frequency) can be added to specify the frequency of the module.



Find inspiration by looking at the first line of an official module tab in the Daemon GUI

¹ Read more in "Start and configure launched modules" chapter in SCANeRstudio_SIMULATION_UserManual.pdf.

IV.F.2 - THE PROCESS CONTROLLER

The Process part of the SCANeR™studio API provides functionalities which allow users to easily manage all the processes aspects of SCANeR™studio. These methods are needed to create a compliant SCANeR™studio module.

A base SCANeR™studio module using the API will look like this “main()” function:

```
int main(int argc, char* argv[])
{
    /*! Initialization of the Process */
    Process_Init(argc, argv);
    /*! Process State */
    APIProcessState status;

    do
    {

        /*! Wait for synchronization at the given frequency */
        Process_Wait();

        /*! Run the process */
        Process_Run();

        /*! Update the status */
        status = Process_GetState();

        if (status == PS_RUNNING)
        {
            // Add user code
        }
    } while (status != PS_DEAD);

    Process_Close();
}
```



In this example the “Process_Init” function is used. This means that this module has to be started with the “-p” and “-c” arguments and the SCANeR API will read these arguments to deduce the configuration and process names.

IV.F.3 - THE COMMUNICATION CONTROLLER

The communication controller is the most import part of the API. It allows users to get or send data over the SCANeR™ studio network or shared memory.

IV.F.3.i - Introduction

The communication controller works only when used inside a valid SCANeR™ studio process created with [Process Controller](#).

Using the Communication controller is divided into four parts:

- **Initialization** (Declare Input, Output data and events)
- **Update Datas to read** (Update the data from the network/shm)
- **Get/Set** values (Get access to the data for a data read, or specification of a data for a data to send)
- **Update Datas to write** (Update the data to the network/shm)



If your APIsample is used:

- to **read** data, then it's necessary to **declare the interface as input**. In this case, data represent input of the API sample.
- to **set** data (cockpit acquisition for example), then it's necessary to **declare the interface as output**. In this case, data represent output of the API sample.

By following this way to send data, all the outputs declared in the initialization step are sent on an **update**. To send a single data, you should use:

- **Initialization** (create an output data)
- **Set** values (Use the data)
- **Update** (Update the data to the network/shm)
- **End** (delete the output data)

There are two ways of using the Communication controller:

- **Asynchronous**: The SCANeR™ studio process run at its own frequency and always gets the most recent values for data.
- **Pseudo-Synchronous**: The SCANeR™ studio process still run at its own frequency but is able to get all values of data using the event system.



The “Event system” of Communication controller can be used to implement specific action on Process controller State changes or Message reception.

Communication controller is using “**dataIdString**” to describe location of data. Such strings are built like that: **Source/Group/Message**

- **Source** is the source of the data, It should be:

- Network (for network messages)
 - Shm (for shared memory data)
- **Group** is the class of messages or the name of shared memory.
 - **Message** is the name of data structure to access.

Some examples:

- "**Shm/ModelCabin/CabToModel**" is the CabToModel structure of the ModelCabin shared memory
- "**Network/IVehicle/VehicleUpdate**" is the VehicleUpdate network message of the IVehicle class of messages.

The Lists of SCANeR™ studio Network messages and Shared memory structures are provided in annexes and in the "Shm.html" and "Network.html" documents available in the "doc" folder of the SCANeR API:

1. [Network](#)
2. [SHM](#)

IV.F.3.ii - Event system

The event system allows users to catch events of the simulation. The events are queued during the **Process_Run()** method. When using event system (at most one registerEvent call used) the calling application **must** clear the event queue (using **Com_getNextEvent()**) after each **Process_Run()**.

Typical application looks like:

```
int main(int argc, char* argv[])
{
    /*! Process initialization */
    Process_Init(argc, argv);
    /*! Events Messages */
    Com_registerEvent("Network/IVehicle/VehicleUpdate");
    /*! Events State */
    Com_registerEvent("PAUSE");
    Com_registerEvent("KILL");
    Com_registerEvent("GO");
    Com_registerEvent("LOAD");

    /***** MAIN LOOP *****/
    bool quit=false;

    while (!quit)
    {
        /*! Wait for synchronization at the given frequency */
        Process_Wait();

        /*! Run the process */
        Process_Run();

        Event* event;
```

```

while ( event = Com_getNextEvent() )
{
    EventType evtType = Com_getTypeEvent(event);

    // ===== Check Event of State type
    if (evtType == ET_state)
    {
        // Add code to handle state change events
        // see State Event management Example
    }
    else if (evtType == ET_message)
    {
        // Add code to handle messages
        // see Messages Event management Example
    }
}
...
}

// Terminate the process
Process_Close();
}

```

IV.F.3.ii.a - SCANeR™ studio States management using events

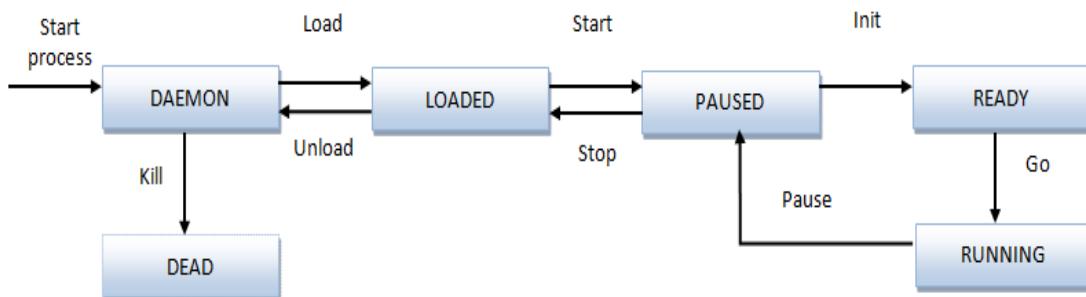


Illustration 5 : SCANeR™ studio state management

Each state change of a SCANeR™ studio API module can be register as a “State Event”. Names of these events are (to use as **Com_registerEvent()** parameter):

- “LOAD”
- “START”
- “INIT”
- “GO”
- “PAUSE”
- “STOP”
- “UNLOAD”
- “KILL”

The type of a State Event is given by the **Com_getStateEventType(event)** method. This method returns an enumerant with possible values:

- ST_Load

- ST_Start
- ST_Init
- ST_Go
- ST_Pause
- ST_Stop
- ST_Unload
- ST_Kill

When registering an State Event the calling application **must validate** the corresponding event by using the method: **Com_validateStateEvent(event)**.

When catching **ST_Load** event (and only with such event) the user can ask

- the scenario name of the load command by using **Com_getScenarioName(event)**;
- the initial conditions by using **Com_getInitConditions(event)**;

Example of code to handle state changes:

```
// ===== Check Event of State type
if (evtType == ET_state)
{
    // Get the State
    StateEventType evtStateType = Com_getStateEventType(event);

    switch (evtStateType)
    {
        case ST_Pause:
        {
            std::cout << " Event : OnPause ! " << std::endl;
            // Add user code
            // Validate the state change
            Com_validateStateEvent(event);
            break;
        }
        case ST_Kill:
        {
            std::cout << " Event : OnKill ! " << std::endl;
            // Add user code
            // Validate the state change Com_validateStateEvent(event);
            break;
        }
        case ST_Go:
        {
            std::cout << " Event : OnGo ! " << std::endl;
            // Add user code
            // Validate the state change Com_validateStateEvent(event);
            break;
        }
        case ST_Load:
        {
            std::cout << " Event : OnLoad ! : " << std::endl;
            char * scenario = Com_getScenarioName(event);
            // Add user code
            // Validate the state change
            Com_validateStateEvent(event);
            break;
        }
    }
}
```



During the ST_Load event, initial conditions of the selected scenario are loaded. If no scenario is opened, the SCANeR API module will return the following error message "setInitialCondition --> Unable to open scenario file".



For performance reason, a SCANeR API module currently loads the first 512 objects attached to the loaded scenario. If more than 512 objects are attached to the scenario the following warning message will appear "ProcessManager::initialConditions --> The maximum number of object 512 has been reached!".

IV.F.3.iii - Network part: Architecture

The aim of the network part is to provide an easy way for client software to get access to some data from the SCANeR™ studio simulation software and send its own data to SCANeR™ studio.

All the data that can be sent/received to/from the network are listed in the document “Network.html” that can be found in the “doc” folder of the SCANeR API delivery.

For instance:

- Accessing data from the interactive or traffic vehicle (corresponding to network messages VehicleUpdate)
- Accessing data that is exchanged by all SCANeR™ studio modules (in particular the Scenario module, controlled by the user) under the form of numerical values through simulation channels (corresponding to network messages “exportChannel”)
- Sending to the SCANeR™ studio modules numerical values through simulation channels, in particular you can send orders to the Scenario module through a specific type of simulation channels (corresponding to network messages exportChannel)
- Sending orders to the Sound module to play sound files
- Sending information messages to ErrorSrv module

IV.F.3.iii.a - Structure

The different SCANeR™studio modules communicate through Ethernet network messages as it is explained in the figure below:

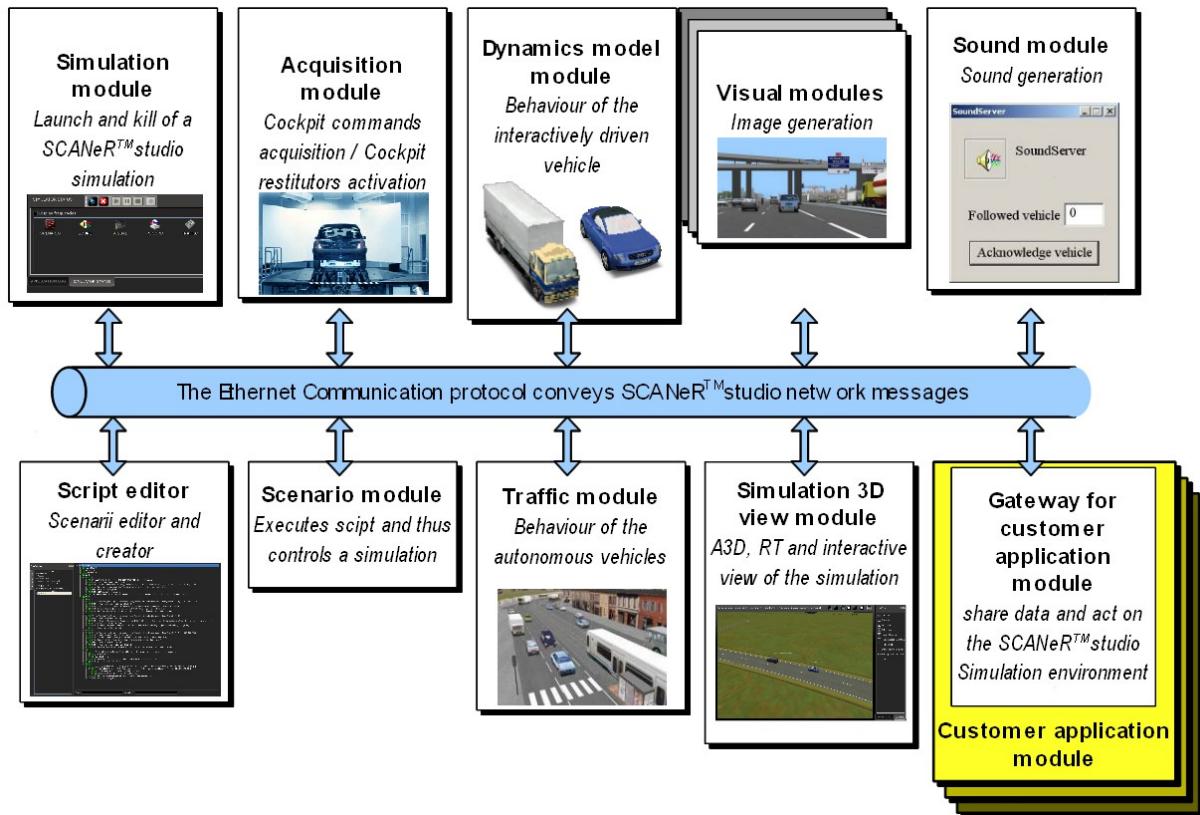


Illustration 6 : SCANeR™studio modules concept

The SCANeR™studio API is a true SCANeR™studio module which can read and send these messages. The client application software, which can be written in any language as it can load DLLs, calls the entry point's functions of the SCANeR API DLL which will trigger the sending and reception of SCANeR™studio Ethernet network messages.

IV.F.3.iii.b - Example

This example shows how to use the COM controller to send informations to the **VISUAL** module

```
int main(int _argc, char* _argv[])
{
    /*! Initialization of the Process */
    Process_Init(_argc, _argv);
    /*! Process State */
    APIProcessState status;
    // Initialization of an interface to send a rain information to the VISUAL module
    DataInterface*myRainInterface=Com_declareOutputData("Network/IWeather/WeatherRain");
    do
    {
        /*! Waiting for timer init (semaphore) */
        Process_Wait();
        /*! Run the process */
        Process_Run();
        /*! Get the process status */
        status = Process_GetState();
        /*! If we are in RUNNING state and after 5 seconds */
        if ((status == PS_RUNNING) && (Process_GetTime() > 5))
        {
            // Ask the VISUAL to add rain
            Com_setFloatData(myRainInterface, "intensity", 0.5);
        }
        // Sending the information on the network
        Com_updateOutputs(UT_AllData);
    }
    while (status != PS_DEAD);
    return 0;
}
```

IV.F.3.iii.c - Network messages with arrays

Some messages use static or dynamic size array of field or structures.

Accessing array values is done by using the following syntax for field name:

- Array of basic field types: **fieldname[index]** (Ex: pos[2])
- Array of structures: **fieldname[index]/structurefield** (Ex: wheelState[2]/speed)

For dynamic sized array a special field “**fieldnameCount**” is used to give the array size. For input messages, this field must be read to know the number of items in the array. For output messages, this field must be filled with the correct number of items.

IV.F.3.iv - SHM part: Architecture

IV.F.3.iv.a - The data Shared Memory

The main purpose of the module is to establish a two way communication between the driver interface – the cabin –, the Dynamics Model (main part and steering part) and Motion Platform Control.

Due to the high frequency required from the model, the shared memory becomes a good replacement of the Ethernet media. It is the heart of the interface.

The following diagram shows communication method between the custom cabin module, the dynamics model and the motion platform control via the shared memory, called SHM which could be accessible by the SCANeR™ studio API.

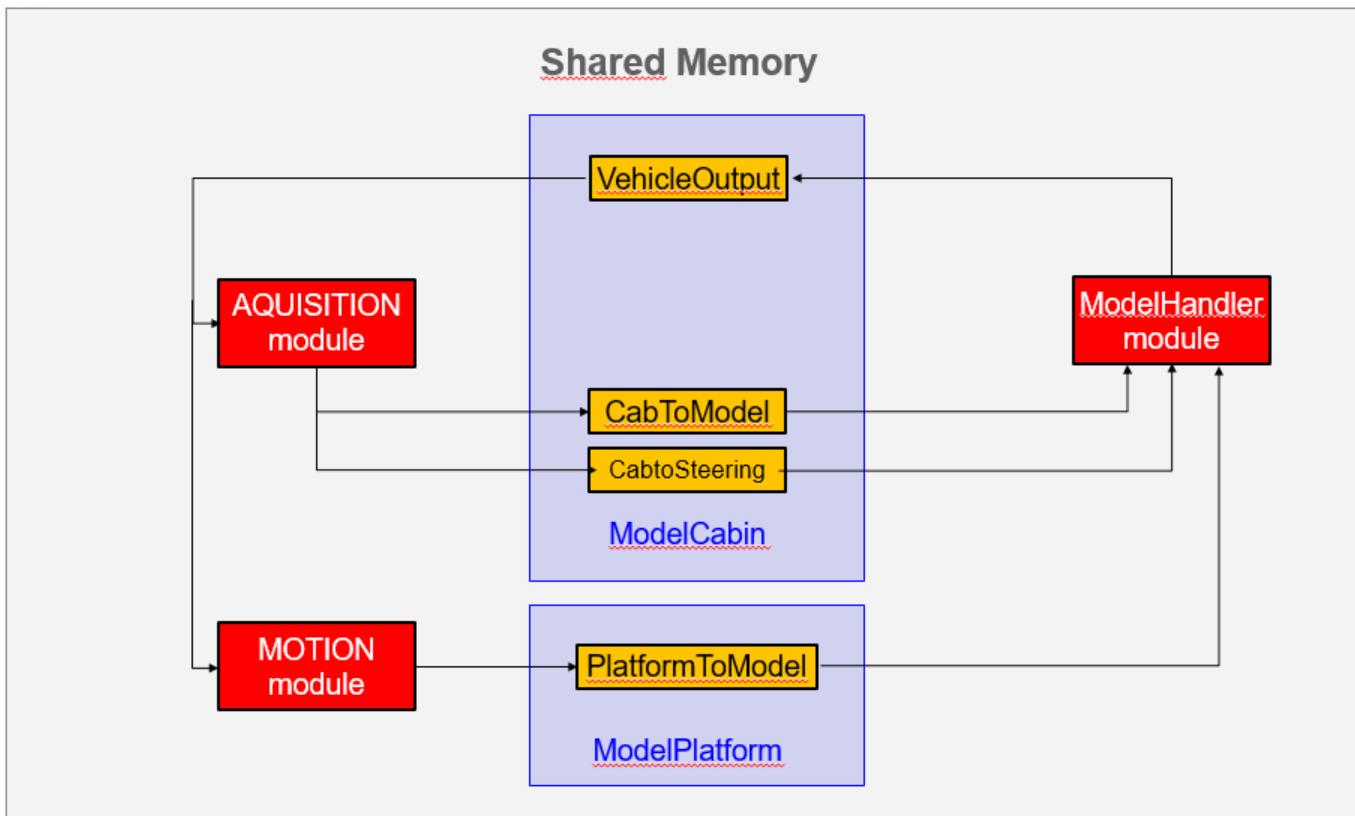


Illustration 7: Shared Memory (Shm)

The main advantages of using SCANeR™ studio API SHM are:

- **Speed:** data shared memory is the best solution for quickly deliver data from a process to another
- **Asynchronous:** data shared memory enables to manage various frequencies of reading and writing.
- **Ease:** data shared memory is very easy to use (no hardware or network dependences).

IV.F.3.iv.b - Vehicle Dynamics Model Input / Output

The vehicle dynamics model outputs are the complete state of the vehicle in the “VehicleOutput” shared memory part.

The Vehicle inputs are divided into two parts:

- CabToSteering: sent from the steering hardware to the vehicle model,
- CabToModel: all remaining vehicle controls (gearbox, pedals, levers etc.) sent by the acquisition module to the vehicle.

Additionaly, when a motion platform is present, the feedback from the platform is written to the "PlatformToModel" shared memory.

IV.F.3.iv.c - Overload the data from Shared Memory

Being able to overload the data from shared memory allows you to take control of the vehicle.

Methods to overload the data from shared memory are named **corrective**.

3 corrective methods are available for ModelCabin interface:

- CabToModelCorrective: Overload CabToModel method (input of ModelHandler) with additive or multiplicative values, to control accelerator, brake, clutch, parking brake, gearbox, gearbox mode.
- CabToSteeringCorrective: Overload CabToSteering method (input of ModelHandler) with additive or multiplicative values, to control steering wheel angle, speed, acceleration, torque.
- SteeringToCabCorrective: Overload SteeringToCab method (output of ModelHandler) with additive or multiplicative values, to control steering wheel angle, speed, acceleration, torque.

The following diagram shows how the corrective methods bypass the standard methods:

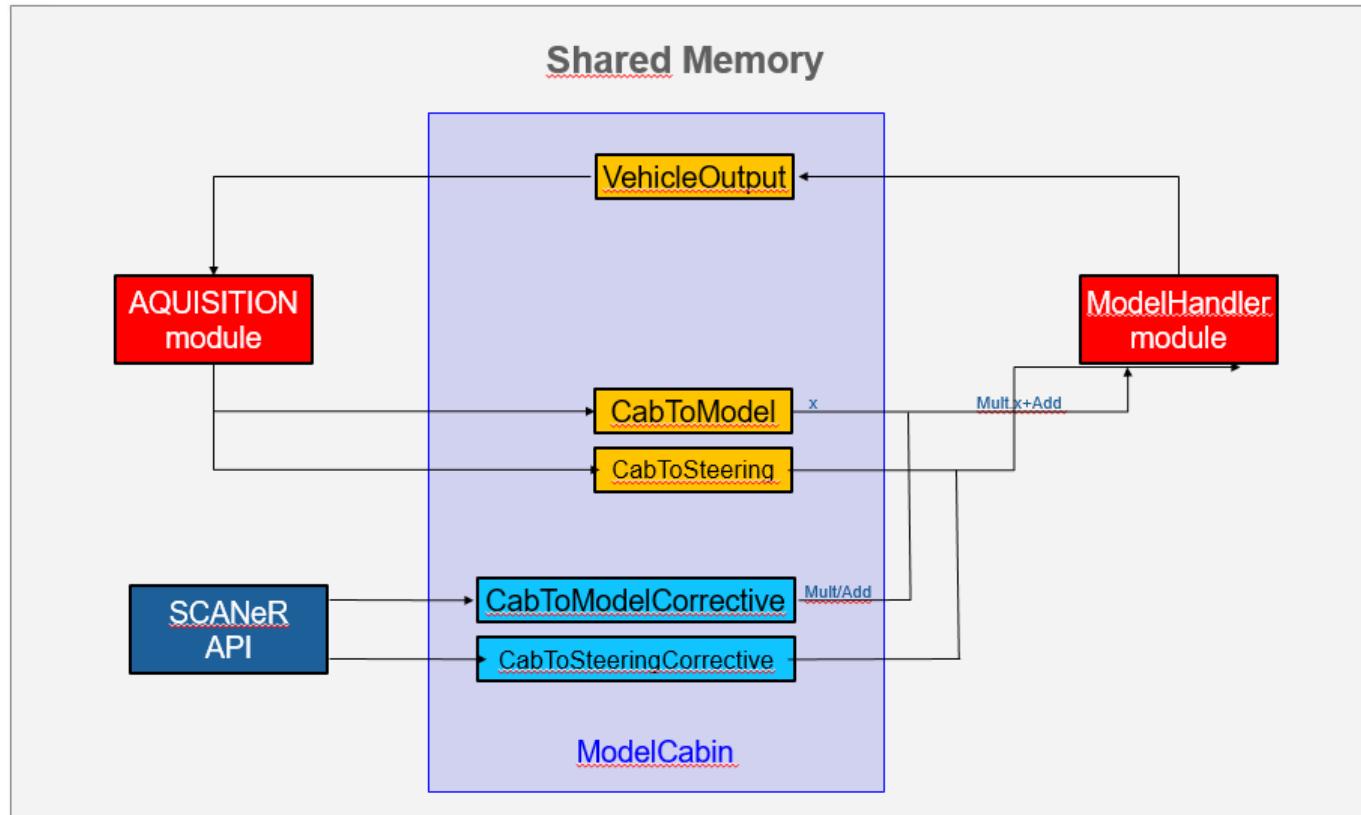


Illustration 8: Shared Memory correctives

When using a corrective method all its fields need to be filled. If you wish to overload only a few part of its values, put a 0 for additive value or\and a 1 for multiplicative value to use the values filled by the non-corrective method. Samples with corrective methods are provided with the installer of SCANeR™ studio in MatLab/Simulink and C++, refer to chapters “IV.J.2.x - SCANeR API TrafficTakeOver - 64“ for C++, “V.G.3 - RVLV - 100”, “V.G.4 - SpeedLimiter - 101” for MatLab/Simulink.



Additive and Multiplicative relation: **Multi.x + Add** (*x is the output value of Cab/SteeringToModel, to do not modify a field Multi must be equal to 1, Add must be equal to 0*),



Full structure of these corrective methods are described into the Shm.html document.



Note MADA: Mada is taking %BrakePedal. But in corrective, you need to correct with Newtons. The conversion taken is: 100% brake pedal pos corresponds at 40daN pedal force.

IV.F.4 - THE SIMULATION CONTROL

The Simulation part of the SCANeR™ studio API provides functionalities which allow users to easily manage all the processes aspects of SCANeR™ studio. It allow the user to create a SCANeR™ studio module.

ProcessInfo data structure is used to get some details about the SCANeR™ studio modules.

- **ProcessInfo.name** : the name of the process.
- **ProcessInfo.state** : the state of the process.

The SCANeR™ studio API Simulation part uses a **configuration file** “launcher.cfg”. It first reads the STUDIO_PATH environment variable. Then retrieve the other configuration files by reading the “datafile.cfg” such as any other SCANeR™ studio module.

- the keyword to retrieve this file in the “datafile.cfg” is :
 - DataConfigLauncher.cfg.
- The syntax is :
 - Scenario0 = Scenario0_name Configuration_name
 - Scenario1 = Scenario1_name Configuration_name
 - ...
 - Scenarion = Scenarion_name Configuration_name



Scenario names are given without the file format extension .sce !

IV.G - ANNEXE 1: CONTROLLER METHODS

IV.G.1 - PROCESS CONTROLLER METHODS

Function name	Function description
<pre>void Process_InitParams(char* ProcName, char* ConfigName, float Frequency)</pre>	<p>Initialize the SCANeR API process with parameters.</p> <p>Param 1 - <i>ProcName</i> : the name of the process.</p> <p>Param 2 - <i>ConfigName</i> : the name of the config.</p> <p>Param 3 - <i>Frequency</i> : the frequency of the process.</p>
<pre>void Process_Init(int argc, char *argv[])</pre>	<p>Initialize the SCANeR API process with command line parameters (behaviour of SCANeR® modules).</p> <p>Param 1 <i>argc</i> : the number of parameters.</p> <p>Param 2 <i>argv</i> : the parameters array.</p> <p>Supported parameters are:</p> <ul style="list-style-type: none"> → -p : the name of the process. → -c : the name of the config. → -f : the frequency of the process. <p> Since Studio 1.5, Process_Init behaviour has changed. Parameters <i>argc</i> and <i>argv</i> are no more used. The function retrieves the command line directly by using a system function (<i>GetCommandLineW</i>). This change has been made to support Unicode command line. We could have changed the Process_Init signature to accept wide chars, but this would have broken the compatibility with all client code.</p> <p>If your code is playing with the command line (like removing client specific arguments before passing it to Process_Init), this won't work any more. The solution in this case is to extract -c, -p and -f arguments, and to call the Process_InitParams function instead.</p>
<pre>APIProcessState Process_Run()</pre>	Run the process of the SCANeR® API and launch all background tasks of the SCANeR® API. This function has to be called regularly by the client application of the API. Return the current state of the process.
<pre>void Process_Wait()</pre>	<p>Sleep the process of the SCANeR® API. This function is used to schedule the client application of the SCANeR® API. The frequency is given during the initialization.</p> <p>This method also updates the time returned by Process_GetTime.</p>

Function name	Function description
<code>void Process_Close()</code>	<p>Close properly the process of the SCANeR® API.</p> <p>If <code>DataInterface*</code> have been created (using the <code>Com_controller</code>) and not removed (using the <code>Com_releaseInterface</code> function) <code>Process_Close()</code> does it.</p>
<code>APIProcessState Process_GetState()</code>	Return the process state which could be: <code>PS_DEAD</code> , <code>PS_DAEMON</code> , <code>PS_LOADED</code> , <code>PS_PAUSED</code> , <code>PS_READY</code> , <code>PS_RUNNING</code> .
<code>void Process_Output(char* msg)</code>	Output a message to the SCANeR Message Log window.
<code>double Process_GetTime()</code>	<p>Return the simulation time in second. The simulation time is the time since the process is in running state.</p> <p>If the process is in daemon or loaded state, the return time is 0.</p> <p>In pause or ready state the time isn't increasing. The return value is the last simulation time.</p> <p>The first step in the running state is at simulation time = 0. The second is simulation time = 1/frequency, the third is 2/frequency... In real time the frequency can not be guaranteed: it's possible that the simulation time evolves differently, not linearly.</p> <p>The time is updated only when <code>Process_Wait</code> is called.</p>

Table 1: Process controller methods

IV.G.2 - COMMUNICATION CONTROLLER METHODS

Function name	Function description
<code>DataInterface* Com_declareInputData (const char *dataIdString, int index = -1)</code>	<p>Declare the input datas in the SCANeR® API.</p> <p>Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>index</code> : the data index</p>
<code>DataInterface* Com_declareOutputData (const char *dataIdString, int index = -1)</code>	<p>Declare the output datas in the SCANeR® API.</p> <p>Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>index</code> : the data index</p>
<code>bool Com_updateInputs(UpdateType updateType = UT_AllData)</code>	<p>Update the input data in the SCANeR® API.</p> <p>Param 1 - <code>updateType</code> : the updated input data type : Network / Shm</p>
<code>bool Com_updateOutputs(UpdateType updateType = UT_AllData)</code>	<p>Update the output data in the SCANeR® API.</p> <p>Param 1 - <code>updateType</code> : the updated output data type : Network / Shm</p>

Function name	Function description
<code>void Com_releaseInterface(DataInterface* dataInterface)</code>	Delete a data interface. Param 1 - <code>dataInterface</code> : the data interface. If this function is not called for a data interface it is automatically deleted by the <code>Process_Close()</code> function (Process controller).
<code>DataInterface* Com_createOutputDataInterface(const char *dataIdString)</code>	Create an output data interface to send a single data. Param 1 - <code>dataIdString</code> : the data path
<code>bool Com_updateOutputDataInterface(DataInterface * dataInterface)</code>	Update the single output data in the SCANeR API. Param 1 - <code>dataInterface</code> : the data interface.
<code>bool Com_deleteOutputDataInterface(DataInterface * dataInterface)</code>	Delete the single output data interface. Param 1 - <code>dataInterface</code> : the data interface.
<code>const char* Com_getStringData(DataInterface * dataInterf const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>short Com_getShortData (DataInterface * dataInterf, const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>long getLongData (DataInterface * dataInterf, const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>float Com_getFloatData (DataInterface * dataInterf, const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>double Com_getDoubleData(DataInterface * dataInterf, const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>char Com_getCharData(DataInterface * dataInterf, const char *fieldName)</code>	Get the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name
<code>int Com_getShortDataArray(DataInterface* dataInterf, const char* fieldName, short* buffer, unsigned int bufferNum)</code>	Get an array of data in a single call. These functions currently work only for Network data, not for Shm.

Function name	Function description
<pre>int Com_getLongdataArray(DataInterface* dataInterf, const char* fieldName, long* buffer, unsigned int bufferNum)</pre>	<p>dataInterf: the data interface on which to retrieve data fieldName: the data name buffer: a pointer to a caller allocated buffer which will receive the data bufferNum: the number of elements to retrieve</p>
<pre>int Com_getUInt64dataArray(DataInterface* dataInterf, const char* fieldName, unsigned long long* buffer, unsigned int bufferNum)</pre>	<p>Return value: -1 if arguments are incorrect or if the field doesn't exist 0 if the message has not been received yet or is empty >0 on success. In this case the return value is the number of element actually copied</p>
<pre>int Com_getFloatdataArray(DataInterface* dataInterf, const char* fieldName, float* buffer, unsigned int bufferNum)</pre>	<p>For variable size arrays, you can use the following pattern: 1) Call this function with buffer=NULL and bufferNum=0 to retrieve the size of the last received message.</p>
<pre>int Com_getDoubledataArray(DataInterface* dataInterf, const char* fieldName, double* buffer, unsigned int bufferNum)</pre>	<p>2) Allocate a buffer with the correct size. 3) Call this function with buffer!=NULL and bufferNum!=0 to fill the buffer.</p>
<pre>Const char* Com_getStringDataByName (const char * dataIdString, const char * fieldName)</pre>	<p>Get the data value. Param 1 - <i>dataIdString</i>: the data path Param 2 - <i>fieldName</i> : the data name</p>
<pre>short Com_getShortDataByName (const char * dataIdString, const char* fieldName)</pre>	<p>Get the data value. Param 1 - <i>dataIdString</i>: the data path Param 2 - <i>fieldName</i> : the data name</p>
<pre>long Com_getLongDataByName (const char * dataIdString, const char* fieldName)</pre>	<p>Get the data value. Param 1 - <i>dataIdString</i>: the data path Param 2 - <i>fieldName</i> : the data name</p>
<pre>float Com_getFloatDataByName (const char * dataIdString, const char* fieldName)</pre>	<p>Get the data value. Param 1 - <i>dataIdString</i>: the data path Param 2 - <i>fieldName</i> : the data name</p>
<pre>double Com_getDoubleDataByName (const char * dataIdString, const char* fieldName)</pre>	<p>Get the data value. Param 1 - <i>dataIdString</i>: the data path Param 2 - <i>fieldName</i> : the data name</p>

Function name	Function description
<pre>char Com_getCharDataByName (const char * dataIdString, const char* fieldName)</pre>	Get the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name
<pre>int Com_getShortDataArrayByName (const char* dataInterf, const char* fieldName, short* buffer, unsigned int bufferNum)</pre>	Get an array of data in a single call. These functions currently work only for Network data, not for Shm.
<pre>int Com_getLongDataArrayByName (const char* dataIdString, const char* fieldName, long* buffer, unsigned int bufferNum)</pre>	dataIdString : the data path of the interface on which to retrieve data fieldName : the data name buffer : a pointer to a caller allocated buffer which will receive the data bufferNum : the number of elements to retrieve
<pre>int Com_getUInt64DataArrayByName (const char* dataIdString, const char* fieldName, unsigned long long* buffer, unsigned int bufferNum)</pre>	Return value : -1 if arguments are incorrect or if the field doesn't exist 0 if the message has not been received yet or is empty >0 on success. In this case the return value is the number of element actually copied
<pre>int Com_getFloatDataArrayByName (const char* dataIdString, const char* fieldName, float* buffer, unsigned int bufferNum)</pre>	For variable size arrays, you can use the following pattern: 1) Call this function with <code>buffer=NULL</code> and <code>bufferNum=0</code> to retrieve the size of the last received message. 2) Allocate a buffer with the correct size. 3) Call this function with <code>buffer!=NULL</code> and <code>bufferNum!=0</code> to fill the buffer.
<pre>int Com_getDoubleDataArrayByName (const char* dataIdString, const char* fieldName, double* buffer, unsigned int bufferNum)</pre>	
<pre>int Com_getCharDataArrayByName (const char* dataIdString, const char* fieldName, signed char* buffer, unsigned int bufferNum)</pre>	
<pre>bool Com_setStringData (DataInterface * dataInterf, const char *fieldName, const char *val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setShortData (DataInterface * dataInterf, const char *fieldName, short val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value

Function name	Function description
<pre>bool Com_setLongData (DataInterface * dataInterf, const char *fieldName, long val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setFloatData (DataInterface * dataInterf, const char *fieldName, float val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setDoubleData(DataInterface * dataInterf, const char *fieldName, double val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setCharData(DataInterface * dataInterf, const char *fieldName, const char val)</pre>	Send the data value. Param 1 - <code>dataInterf</code> : the data interface Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setStringDataByName(const char *dataIdString, const char *fieldName, const char *val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setShortDataByName (const char *dataIdString, const char *fieldName, short val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setLongDataByName (const char *dataIdString, const char *fieldName, long val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setFloatDataByName (const char *dataIdString, const char *fieldName, float val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value

Function name	Function description
<pre>bool Com_setDoubleDataByName (const char *dataIdString, const char *fieldName, double val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_setCharDataByName (const char *dataIdString, const char *fieldName, const char val)</pre>	Send the data value. Param 1 - <code>dataIdString</code> : the data path Param 2 - <code>fieldName</code> : the data name Param 3 - <code>val</code> : the data value
<pre>bool Com_registerEvent (const char* eventId, int index = - 1)</pre>	Register event. Param 1 - <code>eventId</code> : the event id string Param 2 - <code>index</code>
<pre>bool Com_unregisterEvent (const char* eventId, int index = - 1)</pre>	Unregister event. Param 1 - <code>eventId</code> : the event id string Param 2 - <code>index</code>
<pre>Event * Com_getNextEvent()</pre>	Get next event
<pre>int Com_getNbPendingEvent ()</pre>	Get number of pending events
<pre>EventType Com_getTypeEvent (Event * event)</pre>	Get the type of the event. Param 1 - <code>event</code> : the event to get type
<pre>DataInterface* Com_getMessageEventDataInterface (Event* event)</pre>	Get the data interface. Param 1 - <code>event</code> : the event
<pre>const char* Com_getMessageEventDataStringId (Event* event)</pre>	Get the data string id. Param 1 - <code>event</code> : the event
<pre>StateEventType² Com_getStateEventType (Event* event)</pre>	Get the state change type. Param 1 - <code>event</code> : the event
<pre>bool Com_validateStateEvent (Event* event)</pre>	Validate the state change for the event. Param 1 - <code>event</code> : the event to validate state change
<pre>const char * Com_getScenarioName (Event* event)</pre>	Get scenario name, from Load events only. Param 1 - <code>event</code> : the event
<pre>const InitialConditions³ Com_getInitConditions (Event* event)</pre>	Get Initial Conditions, from Load events only. Param 1 - <code>event</code> : the event

² StateEventType is described in

`<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\ScanerAPI\scannerAPI_DLL.Enums.h`

Table 2: Communication controller methods

To get or set network fields which are defined as a table value in the interface, we must to use fieldName with its index.

The following code shows how to get speed field (see IVehicle/VehicleUpdate interface) :

```
std::string ModelOutputId = "Network/IVehicle/VehicleUpdate";
dataModelOutput = Com_declareInputData(ModelOutputId.c_str());

Com_updateInputs(UT_NetworkData);

float speed[6];
speed[0] = Com_getFloatData(dataModelOutput, "speed[0]");
speed[1] = Com_getFloatData(dataModelOutput, "speed[1]");
speed[2] = Com_getFloatData(dataModelOutput, "speed[2]");
speed[3] = Com_getFloatData(dataModelOutput, "speed[3]");
speed[4] = Com_getFloatData(dataModelOutput, "speed[4]");
speed[5] = Com_getFloatData(dataModelOutput, "speed[5]");
```

3 InitialConditions is described in
[`<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\ScanerAPI\scannerAPI_DLL.Enums.h`](#)

IV.H - SIMULATION CONTROLLER METHOD

Supervisor aims are to manage modules, launch/stop simulator, play/stop simulation, etc. A supervisor is not intended to be executed in parallel with SCANeRstudio otherwise network conflicts can appear.

Function name	Function description
<pre>int Simulation_InitParams (char* ConfigName, float Frequency);</pre>	<p>Initialize the process with parameters to do a supervisor (ie capability to manage processes). This function is only needed to build a "supervisor". In this case use this function instead of normal Process_InitParams(...).</p> <p>Param 1 - <i>ConfigName</i>: the name of the configuration.</p> <p>Param 2 - <i>Frequency</i>: the frequency of the process.</p>
<pre>int Simulation_Init (int argc, char *argv[]);</pre>	<p>Initialize the process using given parameters from command line to do a supervisor (ie capability to manage processes). This function is only needed to build a "supervisor". In this case use this function instead of normal Process_Init(...).</p> <p>Param 1 <i>argc</i> : the number of parameters.</p> <p>Param 2 <i>argv</i> : the parameters array.</p> <p>Supported parameters are:</p> <ul style="list-style-type: none"> → -p: the name of the process. → -c: the name of the config. → -f: the frequency of the process.
<pre>bool Simulation_Launch();</pre>	<p>Start all modules declared in the configurationManager and that have the "Selected" box checked.</p>
<pre>Bool Simulation_LoadScenario(int timeoutms, const char* scenarioName);</pre>	<p>Load a specific scenario (not the one declared in the file scenario.cfg).</p> <p>Param 1 - <i>timeoutms</i>: time out in milliseconds (default value = 0).</p> <p>Param 2 - <i>scenarioName</i>: the name of the scenario which will be loaded.</p>
<pre>bool Simulation_Play(int timeoutms);</pre>	<p>Switch all modules to RUNNING state.</p> <p>Param 1 - <i>timeoutms</i>: time out in milliseconds</p>
<pre>bool Simulation_Pause(int timeoutms);</pre>	<p>Switch all modules to PAUSE state.</p> <p>Param 1 - <i>timeoutms</i>: time out in milliseconds (default value = 0).</p>
<pre>bool Simulation_UnLoad(int timeoutms)</pre>	<p>All processes will go to DAEMON state</p> <p>Param 1 - <i>timeoutms</i>: time out in milliseconds</p>
<pre>bool Simulation_Stop(int timeoutms);</pre>	<p>Switch all to LOADED.</p> <p>Param 1 - <i>timeoutms</i>: time out in milliseconds (default value = 0).</p>
<pre>void Simulation_KillAllProcesses();</pre>	<p>Exit all modules.</p>
<pre>bool Simulation_AllProcessesOk ();</pre>	<p>Check if all modules are not in DEAD state. If one process is dead, it calls the launch function to start this one again.</p>
<pre>void Simulation_StartProcess(char* moduleName);</pre>	<p>Start the process with the Id procId.</p> <p>Param 1 - <i>moduleName</i>: the name of the process that needs to be started.</p>
<pre>void Simulation_KillProcess(int procId);</pre>	<p>Kill the process with the Id procId.</p> <p>Param 1 - <i>procId</i>: the process ID that need to be killed.</p>
<pre>void Simulation_UpdateProcessInfo() ;</pre>	<p>Update informations about all modules.</p>

Function name	Function description
<pre>unsigned int Simulation_getProcessNumber();</pre>	Get the number of process informations available
<pre>void Simulation_getAllProcessInfo(APIProcessInfo *array);</pre>	Get all processes informations Pre-allocated array of APIProcessInfo object with the size returned by Simulation_getProcessNumber(). Param 1 - array: Pre-allocated array of APIProcessInfo object with the size returned by Simulation_getProcessNumber().
<pre>int Simulation_GetIdFromName(char *ProcName);</pre>	Get the ID of a particular module with a given name. Module names are declared in the configurationManager. Param 1 - ProcName: The name of the process.
<pre>bool Simulation_GetProcessInfo(int id, APIprocessInfo &inf);</pre>	Get some information about the module associated with the ID id. Param 1 - id: the process id. Param 2 - inf: the process information resulting of the query. <u>APIProcessInfo</u> contains the process: State, Name.
<pre>bool Simulation_GetProcessInfo2(int id, APIProcessInfo2 *inf);</pre>	Get some information about the module associated with the ID id. Param 1 - id: the process id. Param 2 - inf: the process information resulting of the query. <u>APIProcessInfo2</u> contains the process: State, Name, Frequency, Desired frequency and the Hostname.
<pre>bool Simulation_Shutdown(bool allComputers, APIShutdownType shutdownType);</pre>	Shutdown all computers Param 1 - allComputers Param 2 - shutdownType
<pre>const char* Simulation_GetCurrentConfig();</pre>	Get the current configuration name
<pre>int Simulation_GetConfigNumber();</pre>	Get the number of SCANeR configurations available
<pre>const char* Simulation_GetConfigName(unsign ed int index);</pre>	Get a configuration name from its index Param 1 - index
<pre>bool Simulation_ChangeConfig(const char* configName);</pre>	Change the configuration Param 1 - configName: new configuration name

Table 3: Simulation controller method

IV.I - ANNEX 2: LISTS OF DATA

IV.I.1 - NETWORK MESSAGES

See [Network.html](#) (in <STUDIO_PATH>\<STUDIO_VERS>\doc or in the Help / Getting started menu) file for all messages description.

IV.I.2 - SHARED MEMORY

See [Shm.html](#) (in <STUDIO_PATH>\<STUDIO_VERS>\doc or in the Help / Getting started menu) file for all fields description.

IV.J - ANNEXE 3: SAMPLE MODULES IN SCANeR™ STUDIO ENVIRONMENT

IV.J.1 - INTRODUCTION

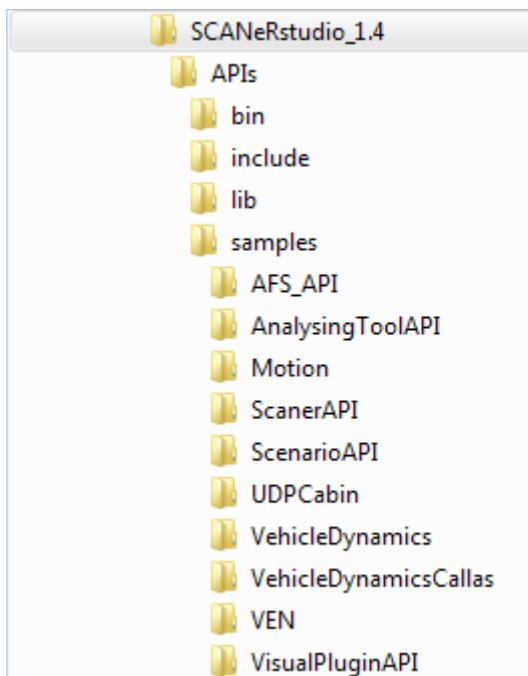


Illustration 9 : Sample Environment

Sample files are in the **samples** folder.

There are 2 ways to launch ScanerAPI Samples:

- ScanerAPI samples can be launched from visual studio. In this case the following arguments have to be mentioned: (Binary file of the Sample Module) -c (Name of the configuration) -p (Name of the Process in the supervisor)

For Ex: scannerAPISampleWeatherControl.exe -c DEMO -p API

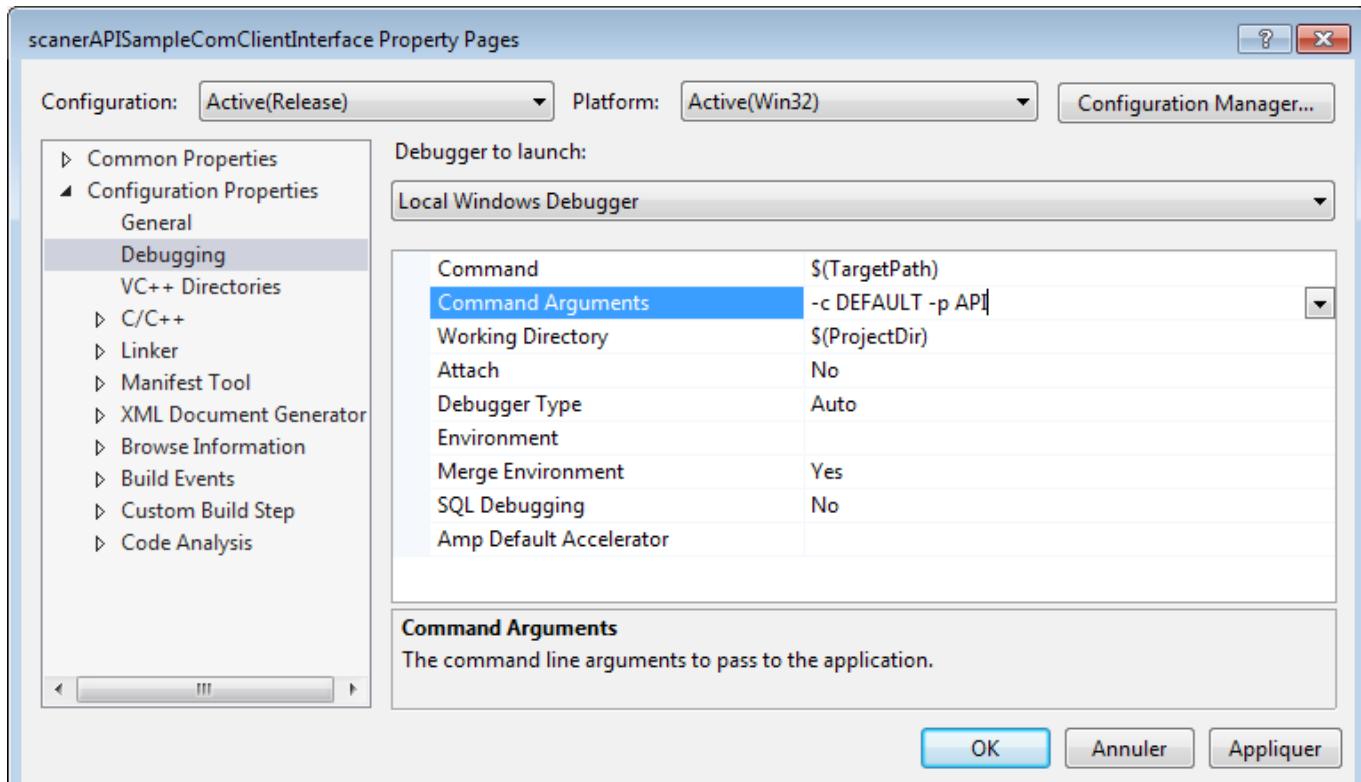


Illustration 10 : Launch a SCAneR API sample from Visual Studio

- ScanerAPI samples can be launched from the SCAneR™ studio Simulation.
 - In this case you need to add a new PROCESS in SCAneR configuration. This new PROCESS will have to use the binary of your sample compiled.
 - Once the configuration has been reloaded the sample can be launched from SCAneR™ studio.

IV.J.2 - DELIVERED SAMPLES DESCRIPTION

IV.J.2.i - SCANeRAPI SAMPLE PROCESS

The aim of this sample is to create a simple SCANeR™ module that only displays its current state.

Sample Process consists of:

- Initializing the process with correct configuration and process name.
- Running the process at each simulation step
- Waiting for the process status to be dead (PS_DEAD state).

IV.J.2.ii - SCANeRAPI WEATHERCONTROL

This sample has been created in order to give an example of the "COM" controller usage associated to a user friendly Graphical User Interface (developed using MFC). Its title is "ControlSFX".

You can add the "Weather control" API module through the configuration manager in Configuration menu. It can be launched by using the minimum following modules.

- **VISUAL**

In this module the SCANeRAPI is used to send network messages (class "Network/IWeather") each time a slider position change.

By "Weather control" module we can control the following parameters:

Parameter	Associated SCANeRAPI Communication Interface	Range
Daytime	Network/IWeather/WeatherDayHour	0 to 24 hours
Fog density	Network/IWeather/WeatherAmbientFog	0 to 500 (and -10 for no)
Rain	Network/IWeather/WeatherRain	0 to 1.0
Sky Saturation	Network/IWeather/WeatherSkySaturation	0 to 1.0
Cloud density	Network/IWeather/WeatherCloudDensity	0 to 1.0
Wind Force	Network/IWeather/WeatherWind (field "speed")	0 to 1.0
Wind Direction	Network/IWeather/WeatherWind (field "direction")	0 to 180 degrees
Snow density	Network/IWeather/WeatherSnow	0 to 1.0
Water accumulation	Network/IWeather/WeatherWaterOnRoad	0 to 20.0 mm
Snow accumulation	Network/IWeather/WeatherSnowOnRoad	0 to 1.0

Tableau 4: Weather Control API

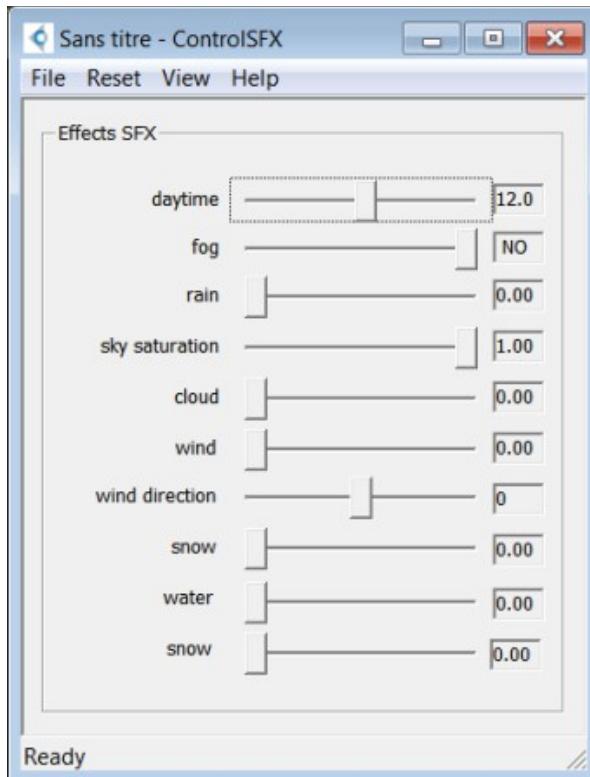


Illustration 11: Weather Control GUI

Some menus are available:

- **File /**
 - **Open:** to restore old settings stored in a file
 - **Save auto:** to save the file
 - **Save As:** to save, in the binary folder, and enter the name of a settings file.
 - **Exit:** to quit
- **Reset:** to set the default values for each sliders
- **View / Status:** to display a status bar that show the module state
- **Help / About:** to show the version

In the schema below we can see that the «WeatherControl» sends the relevant network messages that will be received and taken into account by the **VISUAL** module in order to change the 3D weather environment.

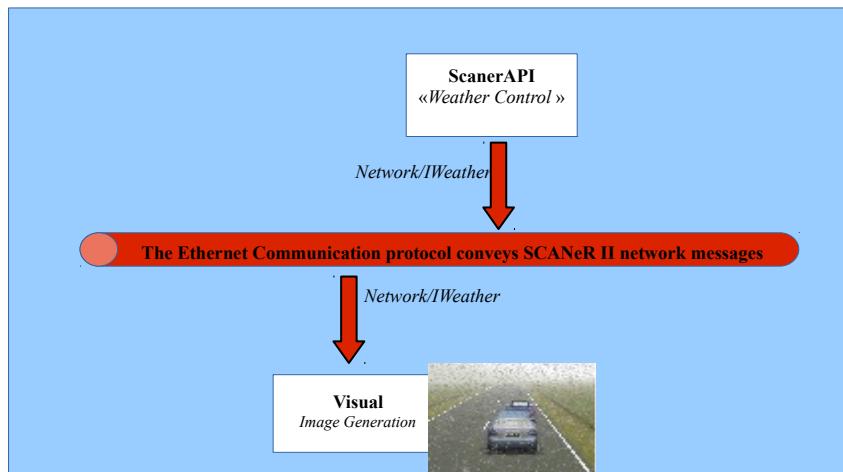


Illustration 12: Weather Control Communication

In "Weather Control" we can control the above environmental conditions while carrying out simulation. For example: by making changes in any of the above conditions during simulation we can analyse the behaviour of driver in medium to extreme conditions while driving in interactive mode and derive the expected data.



The WEATHER dock-widget of the Simulation mode or Scripting Language actions like setFog() can send similar messages.

IV.J.2.iii - SCANeRAPI CLIENT INTERFACE

The SCANeRAPI sample "ClientInterface" is delivered to give an other simple example of the way to use the "Com" controller of the SCANeRAPI in order to send/receive messages to/from the SCANeR™ network.

It can be launched by using the minimum following modules.

- **TRAFFIC**
- **CONTROLPAD**
- **MODELHANDLER**
- **ACQUISITION**

To use this, a scenario with 1 interactive vehicle (idx 0) and 1 autonomous vehicle (idx 1) are needed.

It is used to get and send the following datas from/to SCANeR™ studio network:

Network message	Type	Associated SCANeRAPI Communication Interface
“VehicleUpdate” index 0	Input	NETWORK/IVEHICLE/VEHICLEUPDATE
“VehicleUpdate” index 1	Input	NETWORK/IVEHICLE/VEHICLEUPDATE
“ExportChannel”index 0	Output	NETWORK/IUSER/EXPORTCHANNEL
“ExportChannel”index 21	Output	NETWORK/IUSER/EXPORTCHANNEL
“VisualConstantText Message”	Output	NETWORK/IINFORMATION/SENDVISUALCONSTANTMESSAGE

Tableau 5: Client Interface ScanerAPI

In the schema below “API clientinterface” will receive vehicle update messages for the vehicle with index 0 and for vehicle with index 1, Furthermore this sample send through the network, 2 export channel messages (incremental counter) in order to be received, for example by **CONTROLPAD**. The last function of this module is to send a one shot network message (VisualConstantTextMessage) when a specific condition is reached (Counter == 0) in this case.

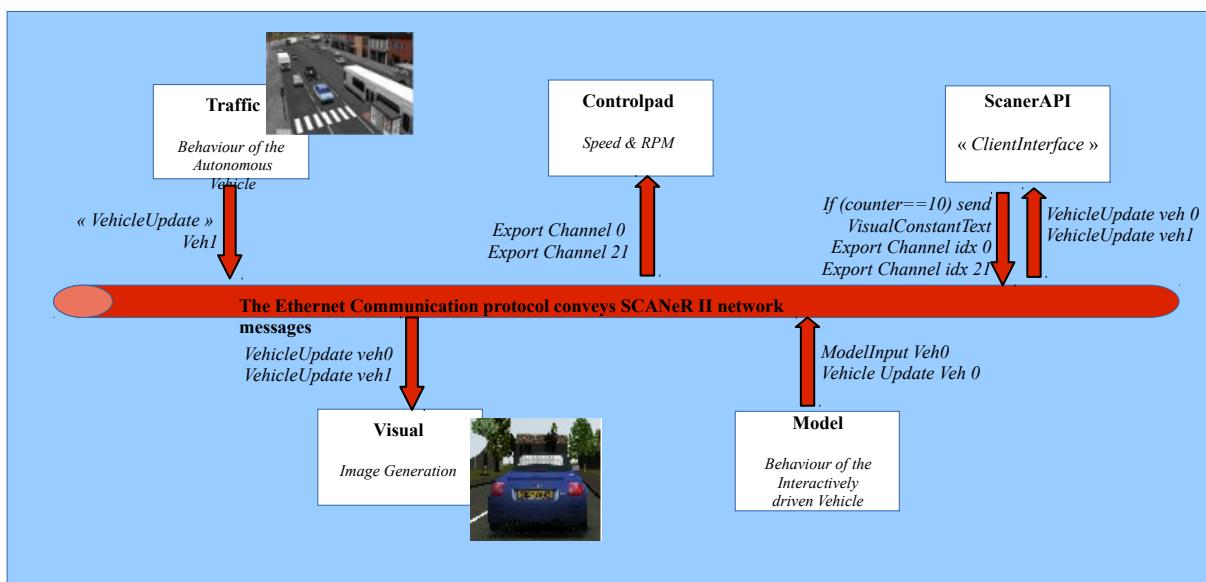


Illustration 13: Client Interface ScanerAPI Communication

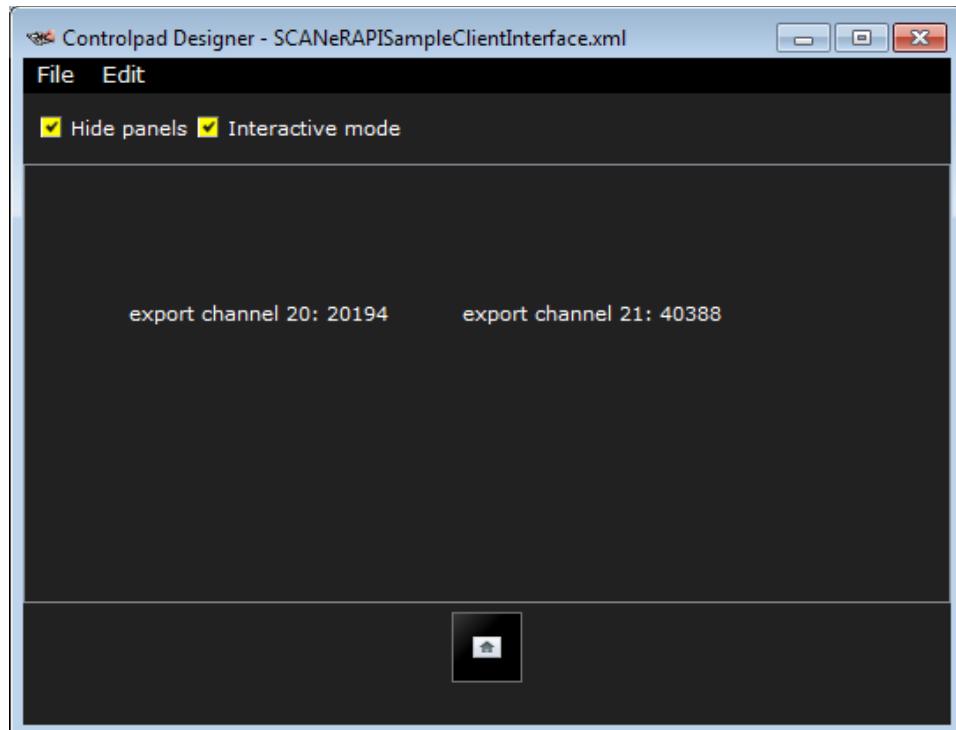


Illustration 14 : Export channel 20 & 21 received in Controlpad

IV.J.2.iv - SCANeRAPI MODEL SHM

The SCANeRAPI sample “ModelSHM” is delivered to give a first simple example of the way to use the “Com” controller of the SCANeRAPI in order to read/write in SCANeR shared memories.

It can be launched by using the minimum following modules.

- **MODELHANDLER**

Shared Memory Structure	Type	Associated SCANeRAPI Communication Interface
Acquisition data given to the dynamic model	Input	SHM/ModelCabin/CabToModel
Output of the dynamic model	Output	SHM/ModelCabin/VehicleOutput

Tableau 6: Model SHM ScanerAPI

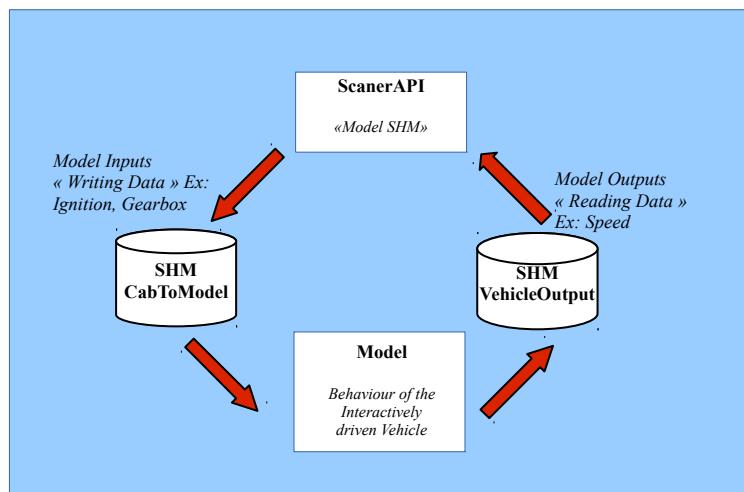


Illustration 15: Model SHM ScanerAPI Communication

It is the data exchange between acquisition and dynamic model. There are two shared memories structures to do these data exchanges.

- CabToModel
- VehicleOutput

IV.J.2.v - SCANeRAPI SCRIPTACQUISITIONMAPPING⁴

The aim of this sample is to get predefined exported channels sent from scripting language using the "COM" controller and to fill the acquisitions SHM in order to give its inputs to the dynamic model. For example use interactive variable to feed the exportChannel() actions.

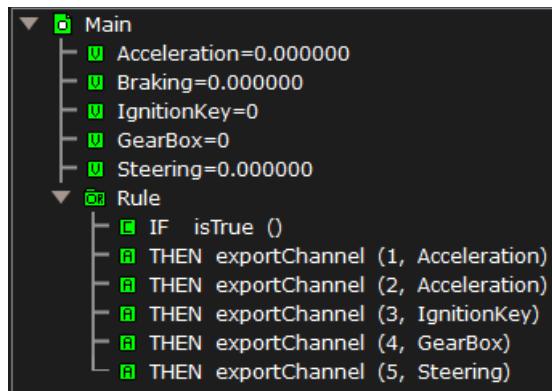


Illustration 16 : scriptAcquisitionMapping MICE

It can be launched by using the minimum following modules.

- MODELHANDLER, SCENARIO

The mapping used is the following:

Export Channel Number	SHM field associated
1	SHM/ModelCabin/CabToModel, field «Accelerator»
2	SHM/ModelCabin/CabToModel, field «BRAKE»
3	SHM/ModelCabin/CabToModel, field «IGNITION_KEY»
4	SHM/ModelCabin/CabToModel, field «GEARBOX»
5	SHM/SteeringCabin/CabToSteering, field «STEERING_WHEEL»

Tableau 7: Mice Acquisition Mapping ScanerAPI

⁴ MICE is the SCANeR™ name for the Scripting Language

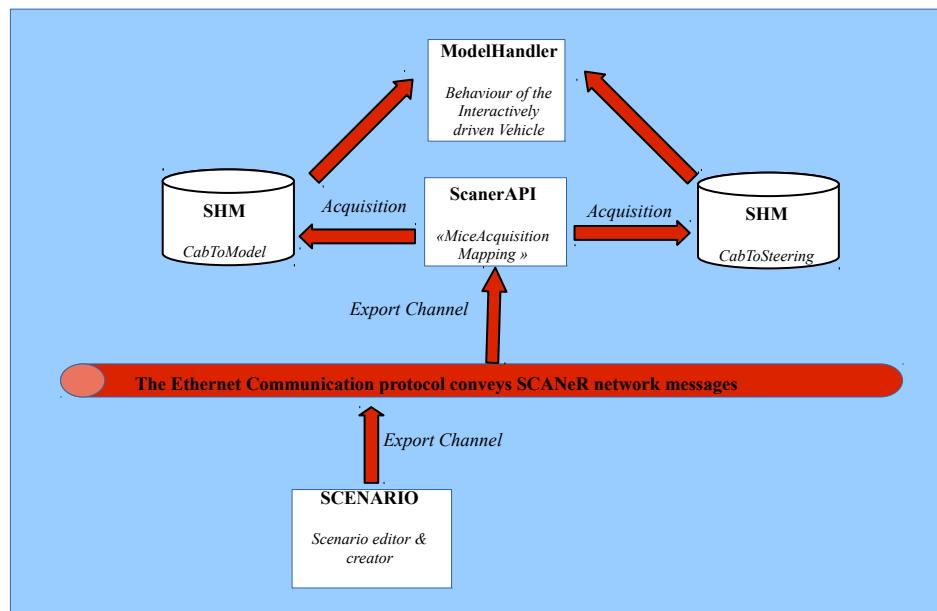


Illustration 17: Script Acquisition Mapping Communication

IV.J.2.vi - SCANeRAPI SPY

It can be launched by using the minimum following modules.

- TRAFFIC

Offers the capability to read every SCANeR™ studio Network messages and/or Shm (Shared Memory) structure

For the most commonly used interface in SCANeR some predefined shortcuts have been defined:

- Vu (Network/IVehicle/VehicleUpdate)
- ec (Network/IUser/ExportChannel)
- ctm (Shm/ModelCabin/CabToModel)

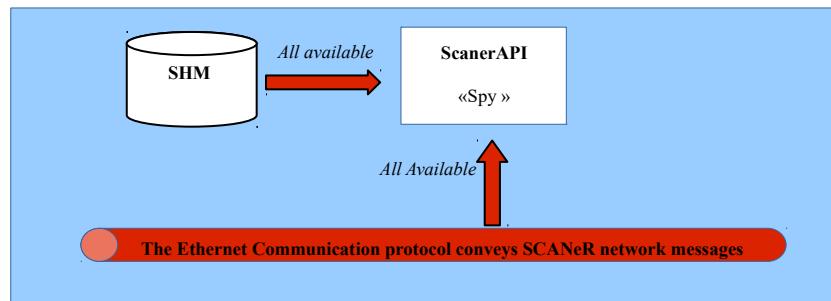


Illustration 18: ScanerAPI Spy Communication

```
c:\Users\gagrwal\ScanerAPI\bin\scannerAPISampleSpy.exe
*****
SCANNER II SPY
*****
*****
```

```
Q:/configurations.cfg
No SCANeR_SWING_USERS environment variable found
No SCANeR_SWING_PATH environment variable found
Default configuration is 222_G2B
Configuration changed to : CONFIG_224
Setting config : [Q:/CFG_USERS/CONFIG_224/data/config/datafile.cfg]
Configration file [Q:/CFG_USERS/CONFIG_224/data/config/datafile.cfg] does not currently handle the possibility to specify the network card to use in a multihomed configuration
Cnet error level 2 : Ask state to 0
Cnet error level 1 : running as process <API>
Cnet error level 0 : SCANeR II API is now alive...

Which message do you want to display ????
Predefined ones : vu <VehicleUpdate>, ni <ModelInput>, mo <ModeOutput>
-> vu

Which index do you want to use ????
Index <example : exportChannel 6, Vehicle 12> =
-> 0

During how many steps ?
Duration <-1 for infinite> :
-> 1
```

```
wheel0 = 0
pos0 = -963.128
pos1 = 113.601
pos2 = 0
angle0 = 0.657987
angle1 = 0
angle2 = 0
speed = 24.9513
turnSpeed = -0.097691
wheelAngle = -0.00986615
state = 0
light = 6
lastUpdate = 11.3005
roadInfo =
wheelState =
```

Illustration 19: ScanerAPI Spy execution Sample

IV.J.2.vi.a - Use case

1. Open a DOS consol

2. go in the folder (by using CD) :
`<STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013`
3. launch : `scancerAPISampleSpy.exe -c <MYCONFNAME> -p SCANERAPISAMPLESPY`
4. launch as explained in `SCANeRstudio_SDK_UserManual` documentation, the **TRAFFIC** module by using the simulation mode
5. create a scenario (with 1 vehicle) where you use, for example, `exportChannel(22, getScenarioClock())` action (so launch **SCENARIO** module too)
6. play the scenario
7. in the DOS console give an answer to the `scancerAPISampleSpy` request (`vu, ec`) : `ec`
8. in the DOS console give an answer to the channel id request : `22` (depends on the example)
9. in the DOS console give an answer to duration request : `40` (depends on what you want)

result => DOS console will show the value (scenario clock) of this channel during 40 steps.

IV.J.2.vii - SCANeRAPI SAMPLE EVENT

It can be launched by using the minimum following modules.

- TRAFFIC

To use this sample, 2 vehicles are needed.

In this sample module, the network messages input interface and the state messages are declared using the "Com_registerEvent" method. The interest of this method is to pop manually EVERY message sent by the other SCANeR modules during the sample last computation step.

Type	Associated SCANeRAPI Communication Interface
Input	NETWORK/IMODEL/MODELINPUT
Output	NETWORK/IVEHICLE/VEHICLEUPDATE

Tableau 8: Sample Event ScanerAPI

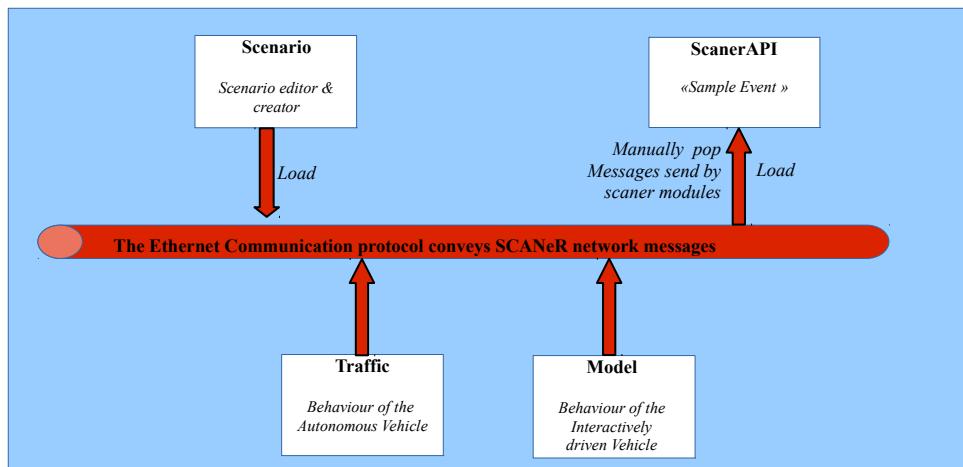


Illustration 20: Sample Event ScanerAPI Communication

IV.J.2.viii - SCANeRAPI RADARDETECTION

This sample shows how to differentiate vehicle types with the script function `getRadarDetection`, it currently differentiates 2 vehicle types : cars and pedestrian.

It uses the network message `SEND_RADAR`, produced by the `getRadarDectection` script function to detect if a vehicle (car or pedestrian) is into the radar beam.

When a car or/and a pedestrian is detected into the radar beam of the `getRadarDetection` script function a sound is played to warn the driver.

To use this sample,

1. Add the SCANeR API module named « `scannerAPISampleRadarDetection` » to your SCANeR configuration and edit its parameters (it is automatically renamed `RADARDETECTION`):

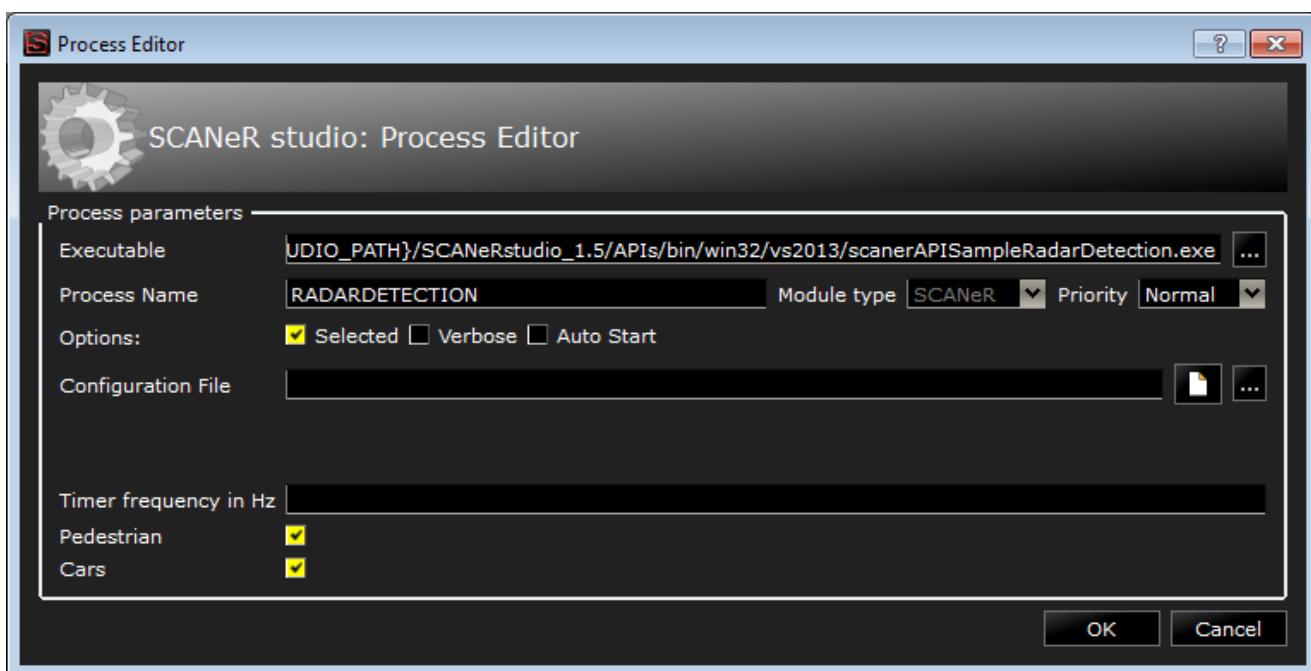


Illustration 21 : RADARDETECTION module configuration

Pedestrian: If checked pedestrian are detected (a sound is played when a pedestrian is in the radar beam), otherwise they are not detected,

Cars: If checked vehicles are detected (a sound is played when a car is in the radar beam), otherwise they are not detected.

2. Create a scenario with:
 - Vehicle ID 0: Interactive vehicle (human driver),
 - Pedestrian and autonomous cars,
 - A script with the function `getRadarDetection` called into an `isTrue` rule. Attach this function to the vehicle ID 0 (refer to the scripting help for `getRadarDetection` details),
3. Run the following modules:
 - `TRAFFIC` (to drive the autonomous vehicles),
 - `RADARDETECTION` (to detect if a target is in the radar beam, if yes, a sound will be played),

- **SCENARIO** (to update the getRadarDetection script function),



Illustration 22 : Sample of a use case of the script function getRadarDetection

- **ACQUISITION & MODELHANDLER** (to drive the interactive vehicle ID 0),
- **SOUND** (to play a sound when a target is detected into the radar beam. A sound is played for a target only if its corresponding option is enabled),
- **WALKERTRAFFIC** (to manage pedestrian).

The result of your simulation depend of your module configuration. Only vehicles who correspond to the checked options will be detected.

IV.J.2.ix - SCANeRAPI RADAR

It is a sample module for Radar detection messages reception using the “COM” controller.

It can be launched by using the minimum following modules.

- **TRAFFIC** (to make vehicle move)
- **SCENARIO** (to set a radar using the scripting function called getRadarDetection)

To use this sample, 1 interactive vehicle and several autonomous vehicle is needed.

Type	Associated SCANeRAPI Communication Interface
Input	NETWORK/IVEHICLE/SENDRADAR

Tableau 9: Radar ScanerAPI

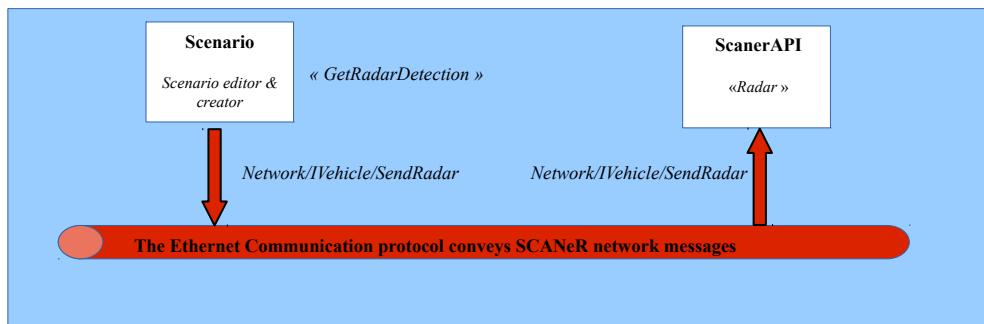


Illustration 23: Radar ScanerAPI - Communication

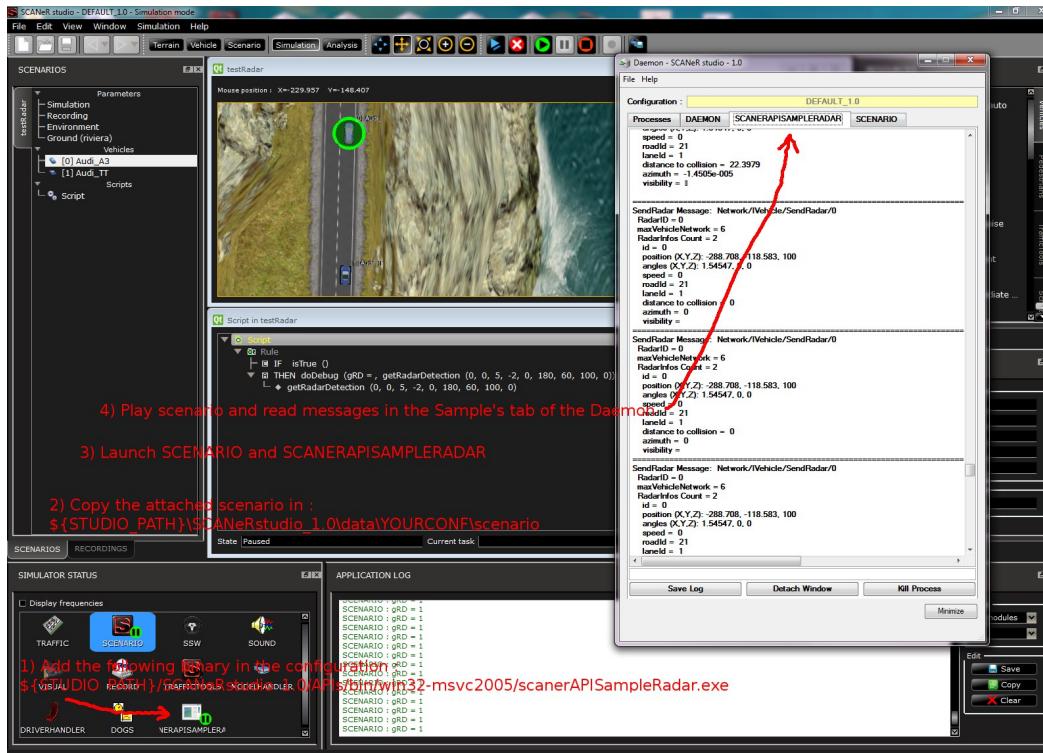


Illustration 24 : Radar ScanerAPI – How to

IV.J.2.x - SCANeR API TrafficTakeOver

This sample demonstrates how to control an autonomous vehicle with the SCANeR API.

To use this sample you need to,

- Build the “scannerAPISampleTrafficTakeOver” sample.
- Add the generated binary file to your SCANeR configuration “**SCANeR API SAMPLETRAFFICTAKEOVER**”
- Open the Studio_TrafficTakeOver scenario.
- Run,
 - **TRAFFIC**, **ACQUISITION**, **MODELHANDLER**, **SCANeR API SAMPLETRAFFICTAKEOVER** and **CONTROLPAD**
- Open **CONTROLPAD** with the “TrafficTakeOver” panel

In this scenario,

- The vehicle id 0 is the MainVehicle, it is a CALLAS vehicle, driven by **TRAFFIC**.
- The vehicle id 1 is the AcquisitionVehicle, it is the same model vehicle as MainVehicle, it is driven by **ACQUISITION**.
- A custom input value can be send from **CONTROLPAD** by using the “TrafficTakeOver” panel to set the driver (traffic or human).
- By default, when you open the scenario for the first time, the driver is the human (Auto Drive button is deactivated into **CONTROLPAD**).
- When the human mode is activated (“Auto Drive” button is pressed), Acquisition are applied from AcquisitionVehicle to MainVehicle, otherwise the vehicle is driven by **TRAFFIC**.

If the “Scenario check results” window appears with the following message: “Custom process **SCANeR API SAMPLETRAFFICTAKEOVER** is associated to some vehicle”, press “OK”.

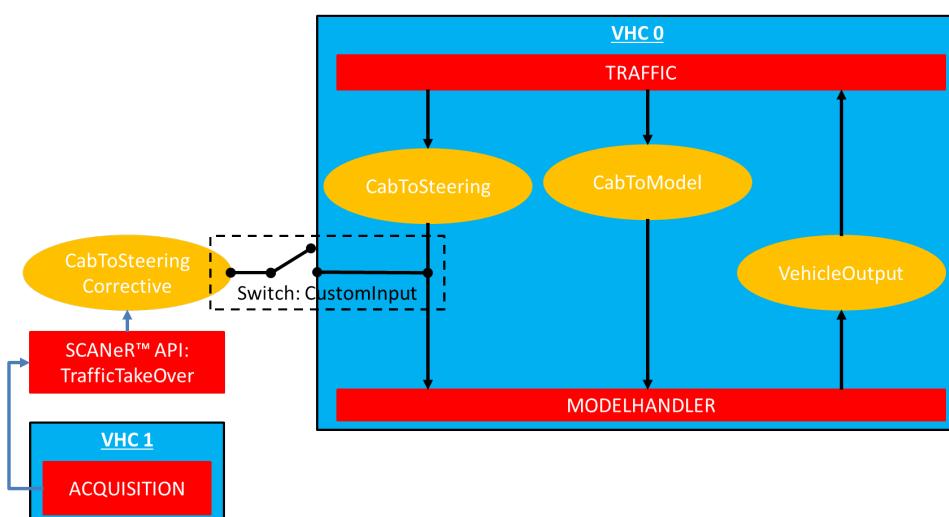


Illustration 25 : TrafficTakeOver Workflow

IV.J.2.xi - SCANeR API ADAS samples

IV.J.2.xi.a - Radar sensor

This sample demonstrates how to detect targets with radar sensors and the SCANeR API. If a target is detected its type is return.

To use this sample you have to create a scenario with:

- Autonomous vehicles (Cars, Bicycles, Pedestrian, Objects).
- The vehicle id = 0 must be equipped with radar sensors (**NOTE**: The vehicle id = 0 can be an Autonomous or a Callas vehicle).
- Others vehicles around the vehicle id = 0 will be used for detection (Cars, Bicycles, Pedestrian, Objects).

Output are displayed into the DAEMON and provided information about the type of the detected target (Ex: "Pedestrian Detected" is displayed if a pedestrian has been detected).

To run this sample you will need modules below:

TRAFFIC + SENSORS + SCANERAPISAMPLEADASRADARTARGETSDETECTION + (Only if use a Callas vehicle) **ACQUISITIONS + MODELHANDLER**

IV.J.2.xi.b - Ultrasonic sensor

This sample demonstrates how to use Ultrasonic sensors with the SCANeR API.

To use this sample you have to create a scenario with:

- 3 vehicles.
- The vehicle id = 0 must be equipped with ultrasonic sensors (Use a Callas vehicle for a better control).
- Other vehicles are used for the detection.

In output a sound is played. If the vehicle id 0 is at:

- the Maximum distance (1 meter) of the others vehicles there is no beep.
- between the Maximum and the Minimum distance ($0,2 < \text{vhc}0 < 1$) of the others vehicles a short beep sound is played.
- the Minimum distance (0,2 meter) with the others vehicles a long beep sound is played.

To run this sample you will need modules below:

SENSORS + SCANERAPISAMPLEULTRASONICPARKINGSENSOR + ACQUISITIONS + MODELHANDLER

IV.J.2.xi.c - Camera sensor

Image Sharing sample shows how to interact with the Image Sharing option available on a Camera sensor.

To use this sample, create a scenario with:

- An autonomous vehicle (Can also be a Callas vehicle if needed).

- This vehicle must be equipped with a CameraSensor with the option “Enable image sharing” checked.

In output a window is created. It reads the Camerasensor shared memory, recreate from the shared memory the Cameraline view and displays into a new window.

To run this sample you will need modules below:

TRAFFIC + CAMERASENSOR + IMAGESHARINGSAMPLE

IV.J.2.xii - SCANeRAPI SAMPLE SIMULATON

The aim of this sample is to give an example of the way to use the "SIMULATION" controller.

Modules defined with «Autostart» option in studio configuration can be launched and act accordingly with the key Input as given below in the table.



Do not use the SCANeRstudio.exe and this sample at the same time.



This is a quick example. You can develop your own controller. It must be compliant with the rules of the state machine described in chapter IV.F.1.i - "State management of the new module" - page 24. Modules must be in the same state, as each other, before initiating a transition.



Launch process before loading and playing the scenario.

Input	Associated Scaner Interface
0	Launch SIMULATION
1	Play SIMULATION (Steps: Init and Go)
2	Pause SIMULATION
3	Stop SIMULATION (from RUNING to LOADED)
4	Unload
5	Load Scenario
6	ALL Process OK?
7	Kill All Process
8	Run
9	Info Process
s	Start Process
k	Kill Process
q	QUIT

Tableau 10: Sample Simulation ScanerAPI

IV.J.2.xiii - SCANeRAPI MATLAB

The aim of this sample is to create a simple SCANeR™ module that displays position of vehicles in a MATLAB plotter.

Sample Matlab consists of:

- Initializing the process with correct configuration and process name.
- Sending data speed, position and orientation at each simulation step
- Waiting for the process status to be dead (PS_DEAD state).

IV.J.2.xiv - SampleMatLabLaserMeter

The aim of this sample is to create a simple SCANeR™ module that gets data and displays ray-tracing detection in a MATLAB plotter.

The sample is here:

```
<STUDIO_PATH>\<STUDIO_VERS>\APIs\ScanerAPI\samples\SampleMatLabLaserMeter
```

This sample can get measures send, by the **LASERMETER** module over the network. The network messages, use methods **LaserMeterHeader** and **LaserMeterPoint** of the Interface **Ivehicle (Network.html)**:

- **LaserMeterHeader**: Anounce arriving results for Laser-meter type sensors:
 - The index of the vehicle
 - The index of the laser-meter
 - The counter of emitted results
 - The number of arriving Laser-meter Points messages (is to size the table bellow):
- **LaserMeterPoint**: Contain results for each laser-meter defined in the LaserMeterHeader
 - The index of the vehicle
 - The index of the laser-meter
 - The counter of emitted results
 - Boolean value true if a point was detected with this ray
 - Horizontal and Vertical angle relative to sensors main direction of this ray
 - Absolute position⁵ (X, Y, Z) of picked point. Valid only if hit is true
 - Relative position⁶ (X, Y, Z) of picked point. Valid only if hit is true
 - Distance from sensor of the picked point. Valid only if hit is true

⁵ In the terrain coordinate system.

⁶ In the vehicle coordinate system.

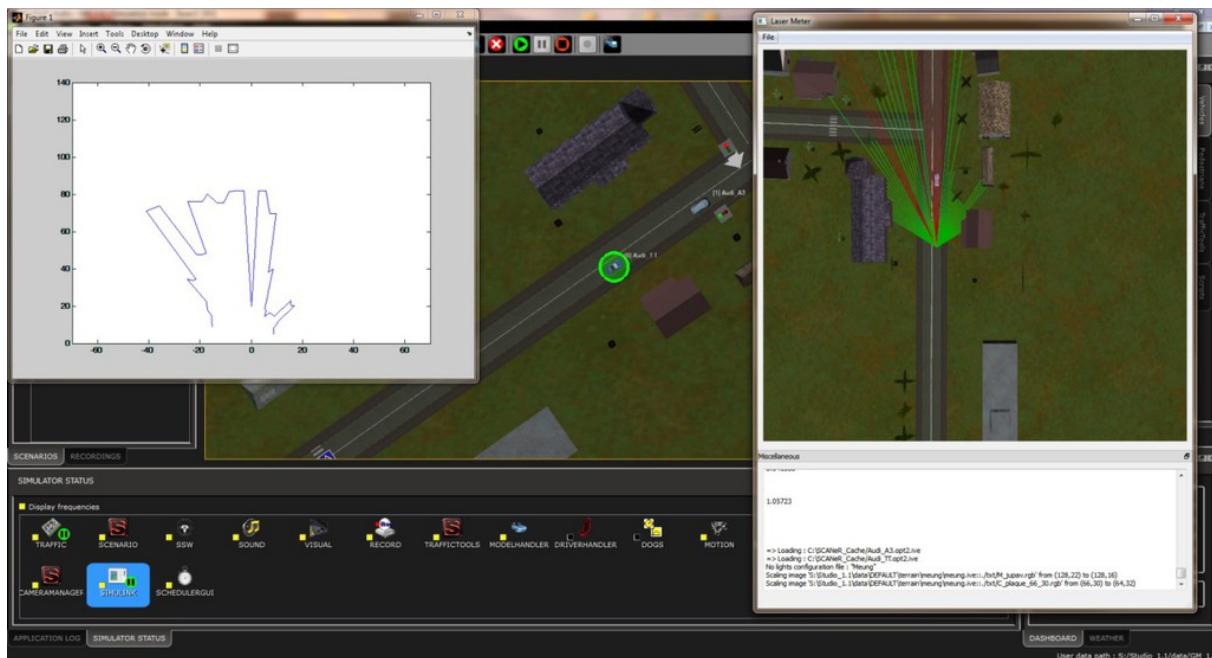


Illustration 26 : SampleMatLabLaserMeter – real time MATLAB processing

Learn more about how to configure and use laser-meter in the related chapter in the [SCANeRstudio_SIMULATION_UserManual](#) documentation.

IV.J.2.xv - ExportChannels sample

The SCANeR API sample “ExportChannels” is delivered to give an example of using the Qt framework⁷ to display messages received from the SCANeR™ network.

It can be launched by using the minimum following modules:

- SCENARIO

To use this, a script with setExportChannel function is needed.

It is used to get and send the following data from/to SCANeR™ network:

Network message	Type	Associated SCANeR API Communication Interface
“ExportChannel” all indexes	In/Out	NETWORK/IUSER/EXPORTCHANNEL

Tableau 11: ExportChannels ScanerAPI

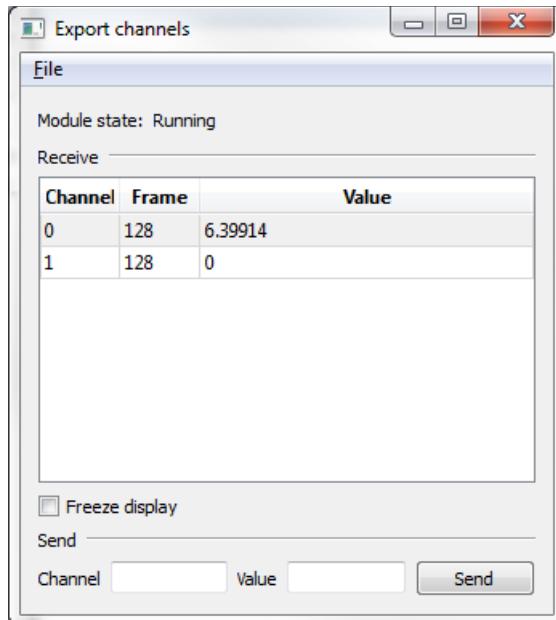


Illustration 27: ExportChannels GUI

All export channel messages sent by other modules are viewed in the Receive list.

The Channel column gives the id of the channel.

The Frame column gives the frame number of the last received message.

The Value column gives the value of the last received message.

If the display is updating too fast, you can freeze the display to examine the values by checking the “Freeze display” check box. The display updating will resume when you uncheck the box.

To send an export channel message to other modules, enter a channel id, a value and click on the “Send” button. Since SCANeR messages are sent to every module but the sender, it is normal that you don't see the sent value in the Receive list.

⁷ An error could occur if "ExportChannels.exe" does not find QTCore4.DLL and QTGUI4.DLL in its folder.

IV.J.2.xvi - ObjectsViewer sample

The SCANeRAPI sample “ObjectsViewer” is delivered to give an example of using the Qt framework to display information concerning the infrastructure objects.

It can be launched by using the minimum following modules:

- PHYSICS

To use this, a terrain with infrastructure objects and/or a scenario with object instances are needed.

It is used to get and send the following data from/to SCANeR™ network:

Network message	Type	Associated SCANeRAPI Communication Interface
Object position computed by PHYSICS	Input	NETWORK_IPHYSICS_INFRAPOSITION
2 objects attachment	Input	NETWORK_IPHYSICS_JOIN
2 objects detachment	Input	NETWORK_IPHYSICS_DISJOIN
Object + vehicle attachment	In/Out	NETWORK_IPHYSICS_JOINVEHICLE
Object + vehicle detachment	In/Out	NETWORK_IPHYSICS_DISJOINVEHICLE
Force an object displacement	Output	NETWORK_IPHYSICS_SETPOSITION

Tableau 12: ObjectsViewer ScanerAPI

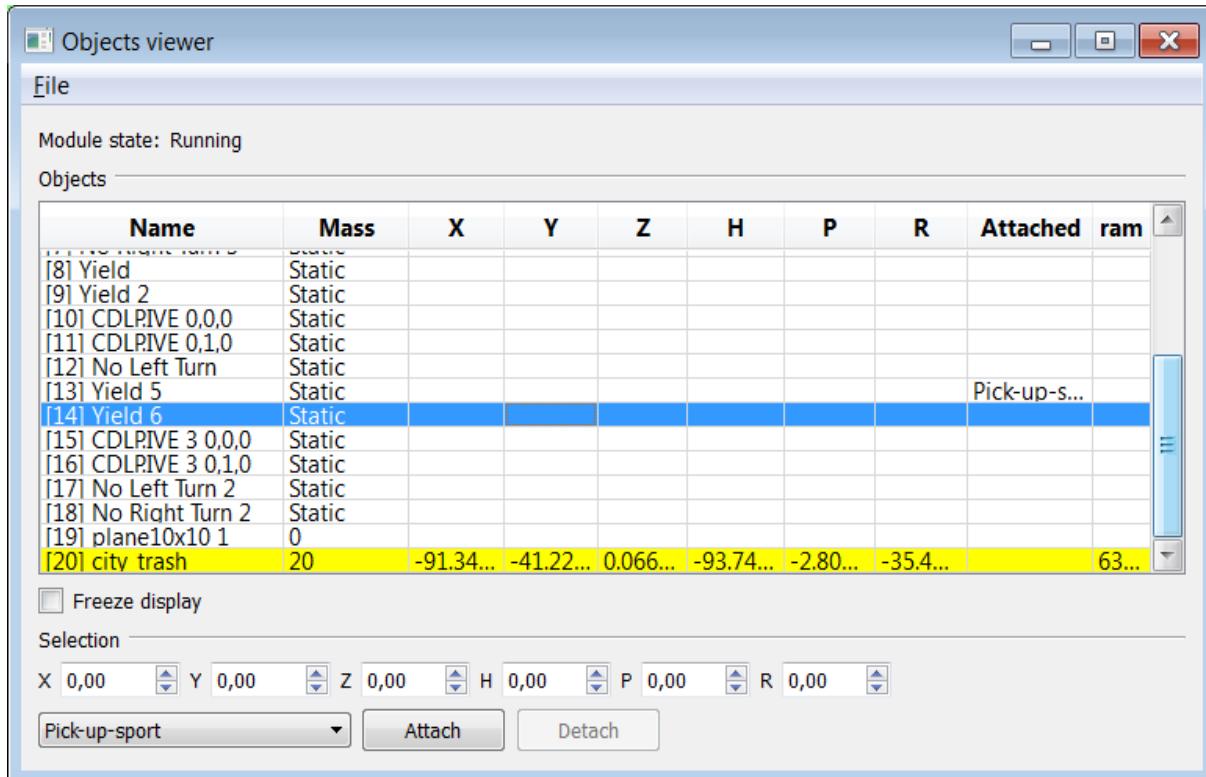


Illustration 28: ObjectsViewer GUI

All terrain objects and object instances of the scenario are shown in the Objects list.

The **Name** column gives:

- the ident of the object into brackets
- the name of the object ; in the case of object sequences, the base name is suffixed by the X index, the Y index and the Z index in the sequence.

The **Mass** column:

- gives the mass of Dynamic objects (i.e. objects you planed to manage with **PHYSICS** module),
- displays 'Static'

The **X**, **Y**, **Z**, **H**, **P** and **R** columns give the position and orientation of the object. It is the initial position until an update message is received. When an object moves, the corresponding line is highlighted in yellow until one second after the move has stopped.

The **Attached** column indicates if the object is attached to a vehicle or to another object.

The **Frame** column gives the frame number of the last received message.

If the display is updating too fast, you can freeze the display to examine the values by checking the "Freeze display" check box. The display updating will resume when you uncheck the box.

If you select an object in the list, you can use the fields at the bottom of the window to move it or to attach / detach it to a vehicle. Note: under certain conditions, attach requests may not be honoured by the **PHYSICS** module. Since there is no way to know if the request was accepted, the object could be wrongly marked as attached in the Objects list.

IV.J.2.xvii - LaserMeterViewer sample

The SCANeRAPI sample “LaserMeterViewer” is delivered to give an example of using the Qt framework to display information concerning the laser meters.

It can be launched by using the minimum following modules:

- LASERMETER

To use this, a vehicle with laser meters is needed.

It is used to get and send the following data from/to SCANeR™ network:

Network message	Type	Associated SCANeRAPI Communication Interface
Laser meter definition	Input	NETWORK_IVEHICLE_LASERMETERHEADER
Laser meter measure	Input	NETWORK_IVEHICLE_LASERMETERPOINT

Tableau 13: LaserMeterViewer ScanerAPI

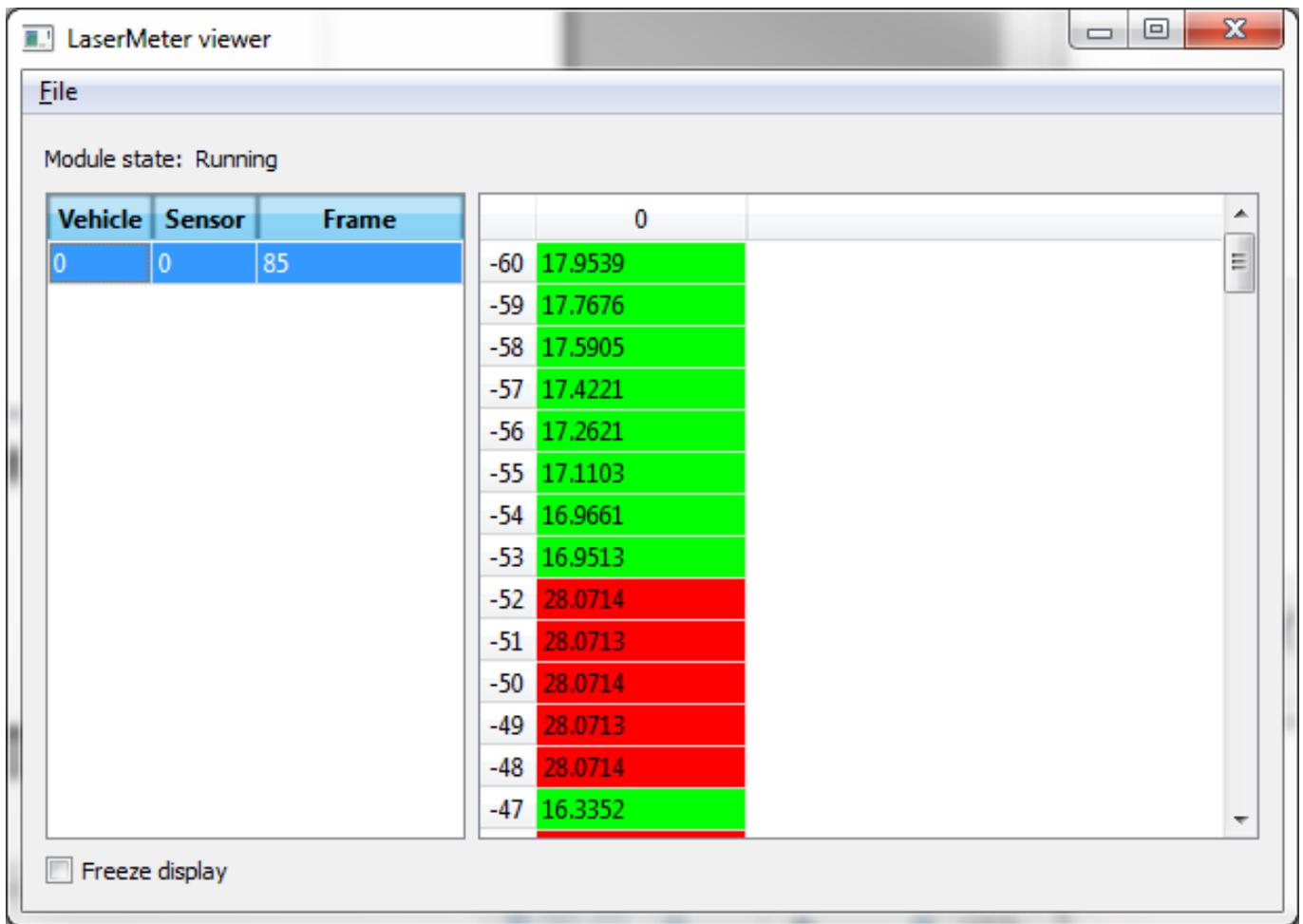


Illustration 29: LaserMeterViewer GUI

All defined laser meters are viewed in the list on the left side.

The Vehicle column gives the id of the vehicle on which the laser is defined.

The Sensor column gives the id of the laser for the owner vehicle.

The Frame column gives the frame number of the last received message.

If the display is updating too fast, you can freeze the display to examine the values by checking the "Freeze display" check box. The display updating will resume when you uncheck the box.

If you select a laser in the left hand list, you can see the measures in the right array. For a display purpose, axes are inverted: columns show the vertical angles, and rows show the horizontal angles. Each cell gives one measure. If the cell is highlighted in green, the value is the distance between the laser and the hit object. If the cell is highlighted in red, the value is the laser range, meaning that no object was hit.

IV.J.2.xviii - SampleDynamicHeightMap

The SCANeRAPI sample “sampleDynamicHeightMap” is delivered to give an example of dynamic_ground use. A dynamic_ground is a type of RESOURCES/Objects the user can drop on the terrain in order to dynamically change the rolling surface).

It can be launched by using the minimum following modules:

- PHYSICS (use de debug option to view objects and animations)

Use the .ObjectsViewer sample to check the ident of the dynamic_ground you want to move (ID of the object to be controlled).

It is used to get and send the following data from/to SCANeR™ network:

Network message	Type	Associated SCANeRAPI Communication Interface
Laser meter definition	Input	NETWORK_IPHYSICS_GROUNDOBJECTALTITUDE

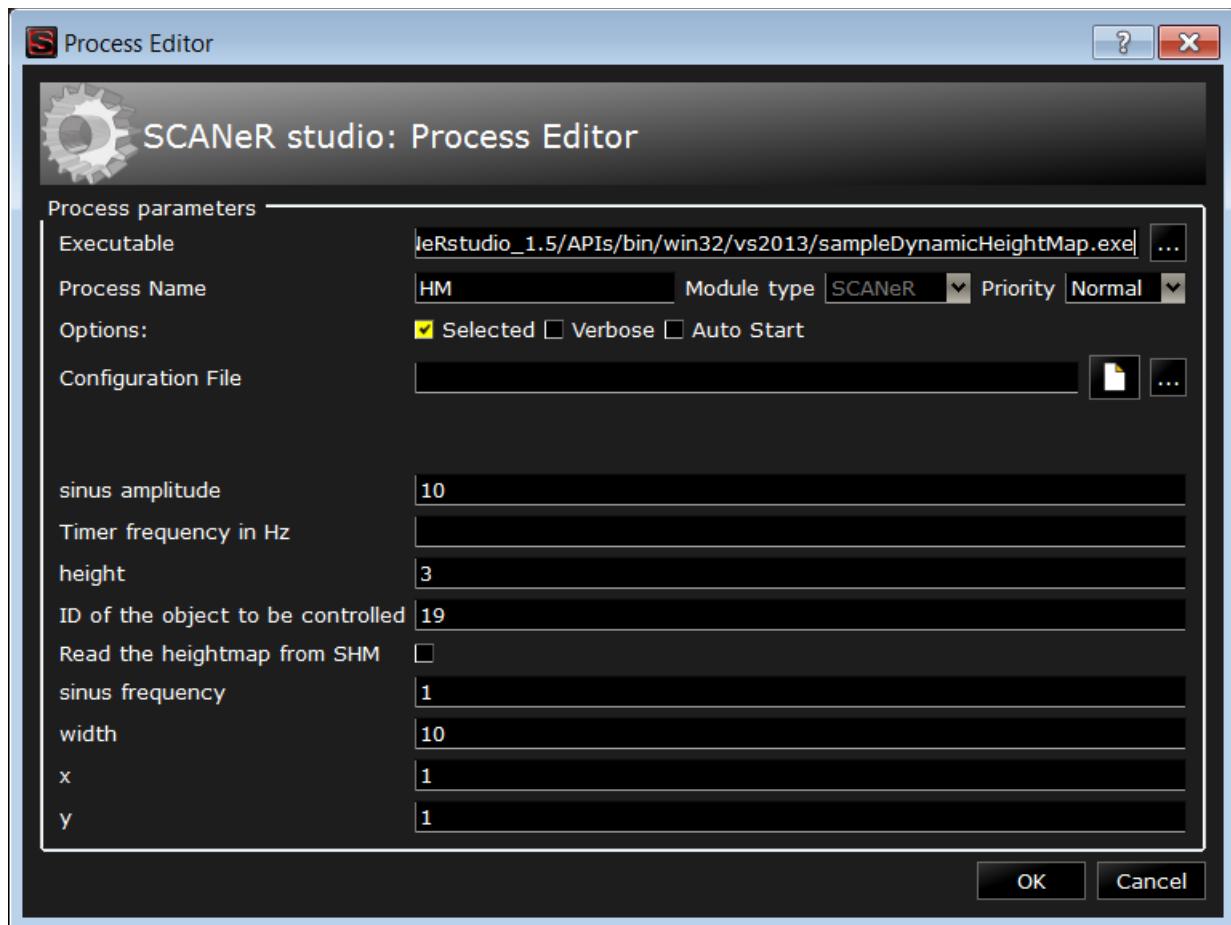


Illustration 30 : DynamicHeightMap

Parameters are:

- sinus amplitude

- height
- ID of the object to be controlled
- Read the heightmap from SHM
- sinus frequency
- width
- x
- y

With the parameters described in Illustration 30 : DynamicHeightMap you can see that a zone of the "plane10x10" dynamic_ground is moving like a wave.

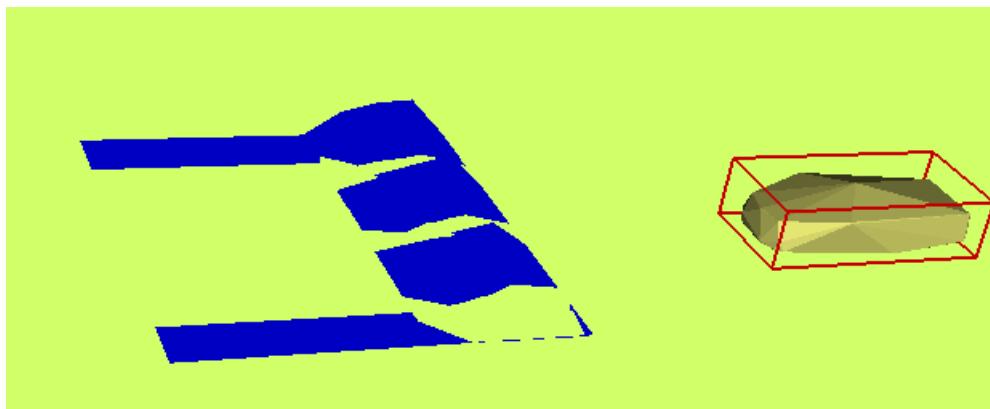


Illustration 31 : the PHYSICS debug option to preview dynamic_ground moves.



There are some limitations as: **VISUAL** does not show such effects, objects falls when they leave the dynamic_ground...

IV.J.2.xix - Python samples

The SCANeRAPI/python samples are delivered to give an example of using the Python scripting language.

It can be used :

1. by installing Python (as described in the IV.B - "Environment" - page 18)
2. by adding adding a module with the parameters below.

- Executable : `c:\python27\python.exe` (or other path you choose when you installed python)
- Extra args : `-u <path_to_the_py_script>`

Remarks:



The current path you could manage in the script is the path where the current `SCANeRstudioDaemon.exe` is running.



Take care about 32/64 bit confusion. Depending on the `python.exe` platform (32 or 64) the current path of the script must be the same as the binary one (32 or 64).

IV.K - PORTING FROM OLD VERSION

The SCANeR API of SCANeR is stable between each release, this means you can keep using the same module (compiled C/C++, LabView or Simulink) without modifications.

Between each major version, modules need to be recompiled and variable naming conventions need to be checked. Some of the shared memory/network packet variables of the SCANeR API used in the module may have changed, this chapter tracks the variable changes between versions to help porting from one version of the ScanerAPI to the other.

IV.K.1 - PORTING SCANeR API FROM 1.4 TO 1.5

IV.K.1.i - Network

1.4			1.5		
Interface	Method	Attribute	Interface	Method	Attribute
IModel	ModelOutput	vhlld	IVehicle	VehicleUpdate	vhlld
		engineStatus	IVehicle	VehicleUpdate	engineStatus
		engineSpeed	IVehicle	VehicleUpdate	engineSpeed
		vehicleStatus	-	-	-
		vehicleStatus was deprecated in 1.4, it has been removed in 1.5.			
		consumption	IVehicle	VehicleUpdate	consumption
		brakesTemperature	-	-	-
			brakeTemperature is available into the Shared memory. If needed, it can be mapped on a custom output to be accessible onto the network.		
		vhlSpeed	IVehicle	VehicleUpdate	speed[0]
		cab_dd_xyzhpr	IVehicle	VehicleUpdate	accel
		accActivated	IModel	ModelInput	RVLVState
		accDriverSpeedSetup	IModel	ModelInput	RVLVSpeed
		gearEngaged	IVehicle	VehicleUpdate	gearEngaged
		robotizedGearBoxMode	IVehicle	VehicleUpdate	GearBoxMode
		heightFront	IVehicle	VehicleUpdate	WheelState[N].posz
		heightBack	IVehicle	VehicleUpdate	WheelState[N].posz

IV.K.2 - PORTING FROM SCANeR API 1.3 TO 1.5

IV.K.2.i - SHM

1,3			1,5		
Interface	Method	Attribute	Interface	Method	Attribute
ModelCabin	ModelToCab	RPM	ModelCabin	VehicleOutput	EngineSpeed
		Speed	ModelCabin	VehicleOutput	cdgSpeed_x
		EngineState	ModelCabin	VehicleOutput	EngineState
		BrakesTemperature	ModelCabin	VehicleOutput	brakeTemperature
		Abs	ModelCabin	VehicleOutput	absIsActive
		RoadTypeFL	ModelCabin	VehicleOutput	laneType
		RoadTypeFR	ModelCabin	VehicleOutput	laneType
		RoadTypeRL	ModelCabin	VehicleOutput	laneType
		RoadTypeRR	ModelCabin	VehicleOutput	laneType
		RoadNatureFL	ModelCabin	VehicleOutput	groundColor
		RoadNatureFR	ModelCabin	VehicleOutput	groundColor
		RoadNatureRL	ModelCabin	VehicleOutput	groundColor
		RoadNatureRR	ModelCabin	VehicleOutput	groundColor
	SteeringToCab	GearEngaged	ModelCabin	VehicleOutput	GearEngaged
		TimeOfUpdate	ModelCabin	VehicleOutput	TimeOfUpdate
	ModelToCabClient	SteeringWheelAngle	ModelCabin	VehicleOutput	SteeringWheelAngle
		SteeringWheelSpeed	ModelCabin	VehicleOutput	SteeringWheelAngle
		SteeringWheelAccel	ModelCabin	VehicleOutput	SteeringWheelAccel
		SteeringWheelTorque	ModelCabin	VehicleOutput	SteeringWheelTorque
		TimeOfUpdate	ModelCabin	VehicleOutput	TimeOfUpdate
ModelPlatform	ModelToPlatform	customOutputValues	ModelCabin	VehicleOutput	CustomOutput
		RoadPitch	-	-	-
		materialId	ModelCabin	VehicleOutput	GroundIndex
		roadType	ModelCabin	VehicleOutput	laneType
		grip	ModelCabin	VehicleOutput	grip
		granularity	ModelCabin	VehicleOutput	granularity
		nature	ModelCabin	VehicleOutput	GroundType
		scannerPos_x	ModelCabin	VehicleOutput	cdgPos_x
		scannerPos_y	ModelCabin	VehicleOutput	cdgPos_y
		scannerPos_z	ModelCabin	VehicleOutput	cdgPos_z
		scannerPos_roll	ModelCabin	VehicleOutput	cdgPos_roll
		scannerPos_pitch	ModelCabin	VehicleOutput	cdgPos_pitch
		scannerPos_heading	ModelCabin	VehicleOutput	cdgPos_heading
		cabAccel_x	ModelCabin	VehicleOutput	cdgAccel_x
		cabAccel_y	ModelCabin	VehicleOutput	cdgAccel_y
		cabAccel_z	ModelCabin	VehicleOutput	cdgAccel_z

1,3			1,5		
Interface	Method	Attribute	Interface	Method	Attribute
		cabAccel_roll	ModelCabin	VehicleOutput	cdgAccel_roll
		cabAccel_pitch	ModelCabin	VehicleOutput	cdgAccel_pitch
		cabAccel_heading	ModelCabin	VehicleOutput	cdgAccel_heading
	StrategyToModel	acceleration_x	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_x
		acceleration_y	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_y
		acceleration_z	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_z
		acceleration_roll	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_roll
		acceleration_pitch	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_pitch
		acceleration_heading	ModelPlatform	StrategyToModel	modulatedVehicleAcceleration_heading
		speed_x	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_x
		speed_y	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_y
		speed_z	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_z
		speed_roll	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_roll
		speed_pitch	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_pitch
		speed_heading	ModelPlatform	StrategyToModel	modulatedVehicleSpeed_heading
		position_x	ModelPlatform	StrategyToModel	modulatedVehiclePosition_x
		position_y	ModelPlatform	StrategyToModel	modulatedVehiclePosition_y
		position_z	ModelPlatform	StrategyToModel	modulatedVehiclePosition_z
		position_roll	ModelPlatform	StrategyToModel	modulatedVehiclePosition_roll
		position_pitch	ModelPlatform	StrategyToModel	modulatedVehiclePosition_pitch
		position_heading	ModelPlatform	StrategyToModel	modulatedVehiclePosition_heading

IV.K.2.ii - Network

1,3			1,5		
Interface	Structure	Attribute	Interface	Structure	Attribute
IVehicle	WheelState	materialId	IVehicle	WheelState	groundIndex
		roadType	IVehicle	WheelState	laneType
Interface	Method	Attribute	Interface	Method	Attribute
INightTest	NIGHT_TEST_SWITCH_PRODUCT	index	INightTest	nightTestSwitchProduct	index
	NIGHT_TEST_SWITCH_PRODUCT_NEXT_OR_PREV	indexOffset	INightTest	nightTestProductOffset	indexOffset

v - SCANNER API WITH SIMULINK

v.A - INTRODUCTION

Since developing driving assistances using the Simulink software is becoming a widespread subject, OKTAL has developed a Simulink library giving access to the SCANeR communication protocol. It's now possible to have access to all the simulation data and to act on simulator components from a Simulink model.

During the design phase, it's easier to do co-simulation. It is possible to start in parallel the SCANeR simulation and the Matlab simulation, both of them communicating together. The Simulink model can then be compiled to an executable module to be used in a production environment on a computer where Matlab is not installed.

The goal of this document is to allow a Simulink user to make a model communicating with SCANeR™studio. Only the specificities of the interface between Simulink and SCANeR™studio are presented in this document. Standard SCANeR™studio functions are presented in other parts of the product documentation.

The following topics are presented:

- Architecture
- First steps
- Model authoring
- Co-simulation
- Code generation
- Samples

V.B - ARCHITECTURE

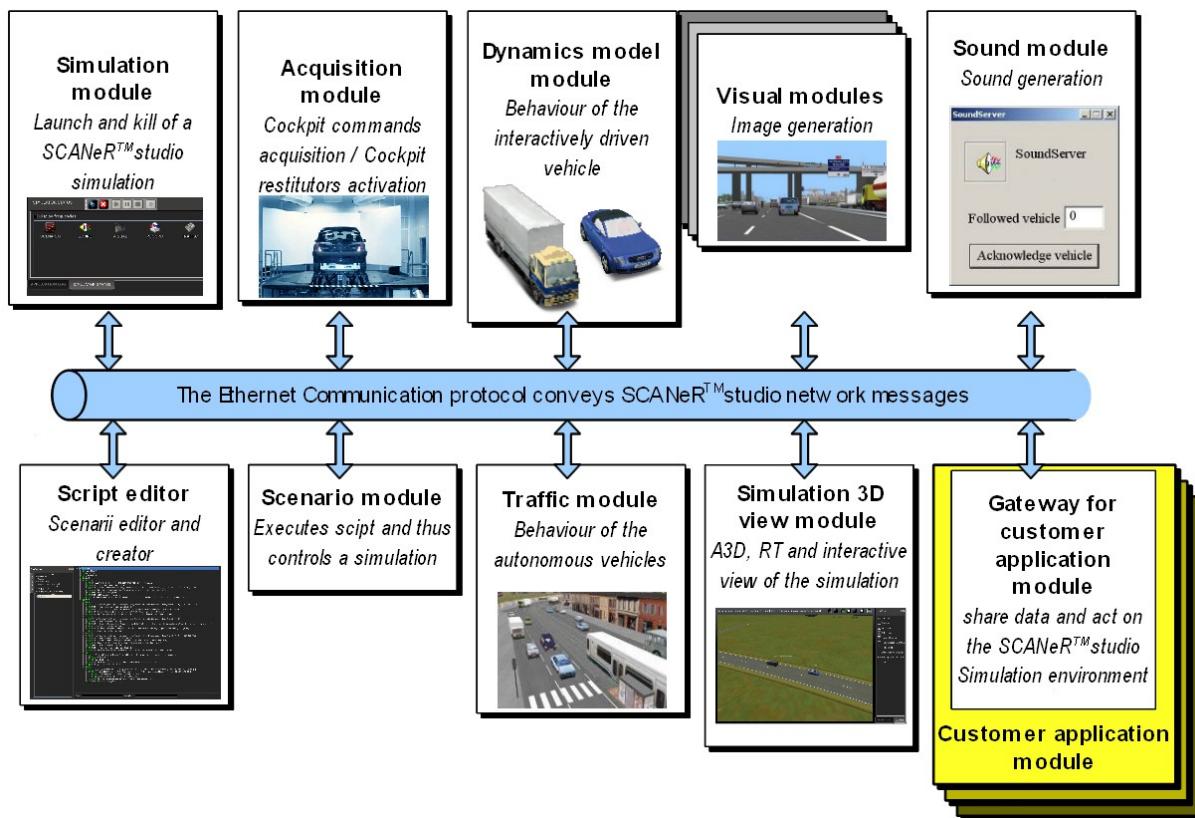


Illustration 32 : Simulink - SCANeR communication protocol

A Simulink S-function developed by OKTAL allows interfacing a Simulink model with SCANeR communication protocols. Like every customer made modules, it is based on the SCANeR API. This S-function reads the network and the shared memory, and sends data to the Simulink model. Similarly, it writes on the network and the shared memory data computed by the Simulink model.

An RTW compilation target allows building an executable module from the model. This executable module is a standalone SCANeR module, which doesn't need any Matlab runtime to run.

V.C - FIRST STEPS

V.C.1 - ENVIRONMENT SETUP

Before using the API, it is necessary to setup the development environment.

V.C.1.i - Matlab path

- From the Matlab main window, make File / Set Path
- Click on Add Folder
- Select that folder: <STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013
- Click on Add Folder

- Select that folder: <STUDIO_PATH>\<STUDIO_VERS>\APIs\simulink\ScanerAPI
- Click on Save and Close

To check that the library has been found in the path:

- From the Matlab main window, click on the Simulink button
- In the library browser, a “SCANeR API” item should be present

v.c.1.ii - Compiler configuration

To build executable code from the Simulink model, it is necessary to have the compiler described in the “Environment” chapter (page 18) installed on the computer. The Matlab command “mex -setup” allows selecting the compiler.

v.c.2 - LIBRARY LAYOUT

The library is organized in 6 categories:

- Root level
- Network inputs
- Network outputs
- SHM inputs
- SHM outputs
- Structures

v.c.3 - SIMULINK BLOCKS

v.c.3.i - Controller

At the root level is found the Controller block, which allows Simulink or the generated code to be detected as a SCANeR module (simulation state management and scheduling).

The Simulink model should contain one and only one Controller block.

v.c.3.ii - Inputs / outputs

Each input or output category contains a subcategory for each message class. Message classes are logical groupings of messages.

Each subcategory contains a block for each message.

An input block is used to retrieve simulation data in the Simulink model.

An output block is used to send Simulink model outputs to other SCANeR modules.

In the Simulink Library Browser, when right clicking on a block and selecting “Help for”, it is possible to access the documentation associated to the block.

V.C.3.iii - Structures

Most signals are of dimension 1, but some of them are structured. It is easier to break down structured inputs into unit signals, and to put unit signals together into structured outputs. To do so, there are a Mux block and a Demux block for each structure.

V.C.4 - COMPILED TARGETS

Two compilation targets are provided: `grt_scannerapi.tlc` and `ert_scannerapi.tlc`. They inherit respectively from `grt.tlc` and `ert.tlc`. Their main purpose is to setup the correct compilation options to build an executable SCANeR module.

When these targets are selected, they also setup some parameters of the Simulink model, which are:

- Fixed step discrete solver
- Infinite stop time
- Continuous time support
- Non-inlined s-functions support
- SCANeR API headers and libraries location

When creating a new Simulink model for SCANeR, it is advised to select the compilation target before any other action. This action allows checking that the model will be compatible with a real time environment, including during a co-simulation.

V.D - MODEL AUTHORING

V.D.1 - COMPILED TARGET

The first step is to select the compilation target:

- Menu Simulation / Configuration Parameters (or CTRL+E)
- Category Real-Time Workshop
- Click on Browse
- Select `ert_scannerapi.tlc` and validate
- Category SCANeR
- Check that the directory is `<STUDIO_PATH>\<STUDIO_VERS>\APIs\simulink\ScannerAPI`
- Validate

V.D.2 - CONTROLLER



Illustration 33: Simulink – Controller block

The second step is to add a Controller block to the model. This one can be found at the root level of the SCANeR API library.

Double click on the block to edit its parameters:

- Configuration: type the name of the current SCANeR configuration name (without forgetting the single quotes). The current configuration name is displayed on the SCANeR™ studio title bar.
- Process name: each module participating in the simulation must have a distinct process name. Type the process name assigned to Simulink (without forgetting the single quotes). This name must also be present in the SCANeR configuration.
- Frequency: execution frequency of the Simulink model.

Note: For the use of the non-real-time mode with Cosimulation (*usage of OFFLINESCHEDULER*) the frequency of the block controller must be **negative** (mandatory). The empty module SIMULINK will display "(S)" to confirm that the synchronization is done:



Illustration 34 : (S) the module is synchronized with Simulink for Cosimulation with the OFFLINESCHEDULER module

- Note: For the use of the non-real-time mode with binary file (.exe) (*usage of OFFLINESCHEDULER*). Next to the binary file, a configuration file must be created: "**ModuleName.exe.res**". To be scheduled, the module's configuration file must contains the following lines:

D:\OKTAL\SCANeRstudio_1.6\APIs\bin\x64\vs2013\ModuleName.exe.res

```

File Edit Search View Encoding Language Settings Macro Run
ModuleKind = Module
StudioOfflineScheduler = Scheduled

```

The screenshot shows a Windows-style text editor window titled "ModuleName.exe.res". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, and Run. The main pane displays two lines of configuration code:

Illustration 35 : Enable Barrier for your binary file with the OFFLINESCHEDULER module

- Network output frequency: to avoid a network overload when executing a model at a high frequency, it is possible to specify a different frequency for network messages sending. This one must be a sub multiple of the model execution frequency.
- Is Synchronized: This can be checked when using the SCANeR module **OFFLINESCHEDULER**. This make the controller wait for the simulation to load before starting. When the option is checked then the Simulink model will wait the SCANeR™ play ("blocking" mode). When the option is not checked the Simulink model will start independently of SCANeR ("unblocking" mode).

The block has two output ports:

- state: simulation state among the following values:
 - 0: Dead
 - 1: Daemon
 - 2: Loaded
 - 3: Paused
 - 4: Ready
 - 5: Running
- clock: simulation time in seconds. This time is reset when starting the simulation and when leaving a pause.

V.D.3 - LAUNCHER



Illustration 36: Simulink – Launcher block

This block is used to control the launch of the SCANeR modules, the loading of a scenario, and the launching of a simulation.

Double click on the block to edit its parameters:

- Configuration: type the name of the current SCANeR configuration name (without forgetting the single quotes). The current configuration name is displayed on the SCANeR™ studio title bar.
- Scenario: type the name of the scenario which will be loaded, without its extension. The scenario must be in the **/Data/Scenario** folder of the configuration set in the module.
- Timeout: Starting modules and loading scenario in SCANeR takes a short time. This timeout allows the Launcher module to wait for a set time before considering SCANeR™ studio didn't launch. The default value is 120,000 milliseconds = 2 minutes.

V.D.4 - NETWORK AND SHM INPUT BLOCKS

Input blocks allow using simulation data in the Simulink model. These blocks have output ports but don't have input ports. They are orange colored. Their default name is made up with the category and the message name followed by "Input". It is possible to rename the block to give it a more significant meaning depending on its role in the model. The text inside the block will still display the category and the message name.

Each output port represents a field of the message. When a message is received, all fields are updated at the same time step. You just have to connect output ports to other Simulink blocks to use the data in the model.

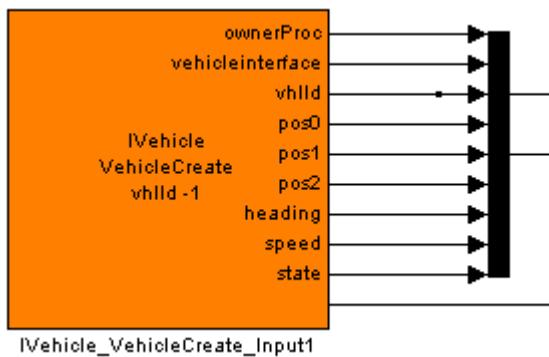


Illustration 37: Simulink - Input block

V.D.5 - NETWORK AND SHM OUTPUT BLOCKS

Output blocks allow sending Simulink model outputs to other SCANeR modules. These blocks have input ports but don't have output ports. They are blue colored. Their default name is made up with the category and the message name followed by "Output". It is possible to rename the block to give it a more significant meaning depending on its role in the model. The text inside the block will still display the category and the message name.

Each input port represents a field of the message. A message is always sent with all of its fields. It is important to fill all fields in order not to send incoherent values to other modules. You just have to connect other Simulink blocks to the input ports to send model data.

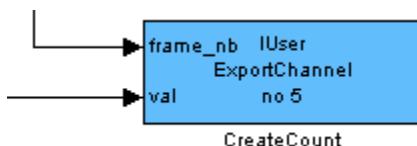


Illustration 38: Simulink - Output block

V.D.6 - SHM BLOCKS

These blocks allow reading or writing in the shared memory. The shared memory is used by some modules to establish a high performance communication with **MODELHANDLER**. To read or write in the shared memory, Simulink or the RTW generated executable must run on the same computer than **MODELHANDLER**.

To have the possibility to run several **MODELHANDLER** processes on the same computer, shared memory blocks have an index. To communicate with the desired vehicle, the SHM index of the block must be configured correctly. To do that, double click on the block to edit its parameters and type the index corresponding to the vehicle identifier as it is displayed in the scenario (between brackets on the left side of the name). After validating, the block displays the chosen index.

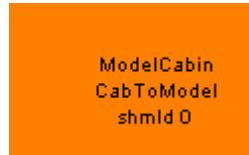


Illustration 39: Simulink – SHM block

V.D.7 - NETWORK BLOCKS

These blocks allow exchanging network messages with other SCANeR modules. Since messages are sent using a multicast address, Simulink or the RTW generated executable can run on any computer participating in the simulation.

For input blocks, as long as no message has been received, all output ports have a zero value. After at least one message has been received, output ports have the value of the last received message. If the Simulink model runs faster than the module sending the message, several model steps will be done with the same values for this block outputs. If the model runs slower, it will have a subsampling of the values.

For output blocks, a message is sent at the specified output frequency, even if the values computed by the model didn't change. However, it is possible to send a message only on a value change by putting the block in a triggered subsystem.



Illustration 40: Simulink – Network block

V.D.7.i - Indexed messages

Some messages represent a unique data and can't be indexed. For example, the `IWeather_WeatherRain` message represents the rain intensity in the simulation.

Other messages represent the state of a particular object and can be indexed. For example, the `IVehicle_VehicleUpdate` message represents the position of one vehicle. To communicate with the desired vehicle, the block index must be set correctly. To do that, double click on the block to edit its parameters and type the index corresponding to the vehicle identifier as it is displayed in the scenario (between brackets on the left side of the name). After validating, the block displays the chosen index.

To take another example, the `IUser_ExportChannel` message allows reading or writing to an export channel. The message index corresponds to the channel number.

For output blocks; it is possible to dynamically change the index, if this one is computed by the model. To do that, set the index parameter to -1. After validating, an extra input port appears. Due to a limitation of Matlab 2008, the name of this extra port is not displayed. The value received on this port will be used as the index when sending the message.



Illustration 41: Simulink – block index

V.D.7.ii - Using events for input blocks

Some messages represent the current state of an object. For example, the `IVehicle_VehicleUpdate` message represents the current position of a vehicle.

Other messages don't represent a state but are used to send a command from one module to another. For example, the `IVehicle_VehicleCreate` message allows dynamically creating a vehicle. This message has a meaning only at the time it is received. The model must then know at each time step if the block output data is representing a new message or is the one of an already processed message. To do that, double click on the block to edit its parameters and check the Use events checkbox. After validating, an extra output port appears. Due to a limitation of Matlab 2008, the name of this extra port is not displayed. The value of this output port is equal to 1 at the time step following the message reception, and 0 otherwise.



Illustration 42: Simulink - event

For indexed messages, it is possible to specify an index of -1. In this case, all messages will be taken into account and will generate an event. It is possible to know the index of an event by using the corresponding output port.

It is not possible to specify an index of -1 without using events. In effect, the block would take into account all messages independently of their index and would output the content of the last received message. It would then be impossible to retrieve all messages independently.

When using events, it is advised to run the Simulink model at a high frequency. Indeed, the model can process only one message at each time step. If messages are sent at a frequency higher than the one of the model, the messages will accumulate and will lead to memory saturation and time delay in input data processing.

An input block with events enabled must not be put in a conditional subsystem. Indeed, event management is global to the whole model. All events are put in the same queue and are delivered to the corresponding block in their arrival order. If the first message of the queue is for a block placed in an inactive subsystem, it will forbid all other blocks to receive their messages.

V.D.7.iii - String fields

Some messages have string fields. It is not possible to exchange string between Simulink blocks.

For input blocks, string fields are not visible.



For output blocks, string fields will be static. They **MUST** be set in the block parameters dialog.

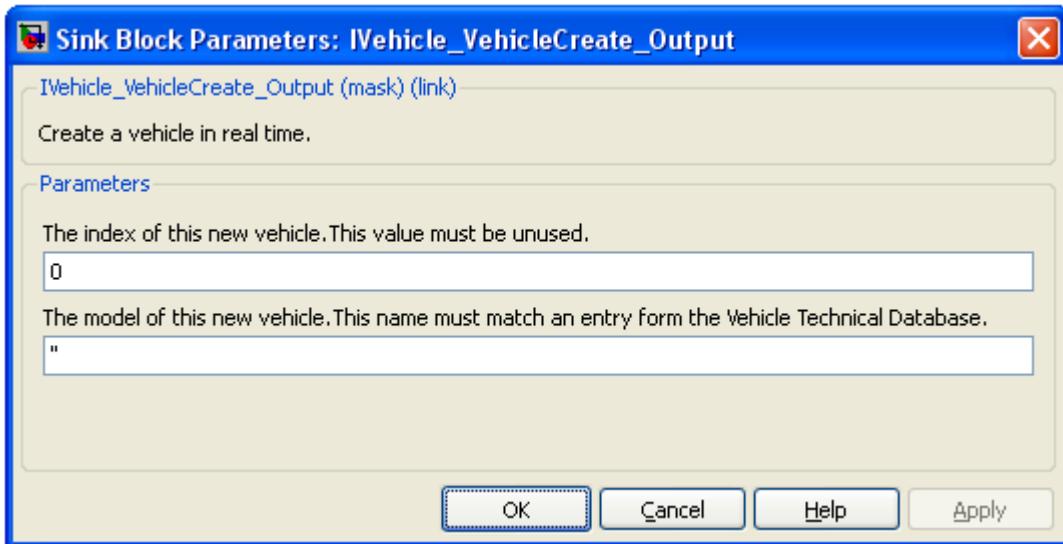


Illustration 43: String fields

V.D.8 - UNIQUENESS

It is not possible to have more than one block with the same triplet (direction, message, index). If it's the case, Simulink will show an error message when starting the co-simulation or when generating the code.

V.D.9 - ARRAYS AND STRUCTURES

Most signals are of dimension 1, but some of them are arrays or structures. When it's the case, the signal name displayed in the block is followed by its type and its dimension. A star indicates a dynamically sized array. In this case, the signal just above is a Count signal indicating the number of elements in the array.

V.D.9.i - Fixed size arrays



Illustration 44: Simulink - Fixed size array

For an input block, the array field output port must be connected to the input of a Demux block which has a number of outputs equal to the displayed dimension. All Demux block outputs will then have a dimension of 1.

For an output block, the array field input port must be connected to the output of a Mux block which has a number of inputs equal to the displayed dimension. All Mux block inputs must have a dimension of 1.

V.D.9.ii - Variable size arrays



Illustration 45: Simulink - Variable size array (Input block)



Illustration 46: Simulink - Variable size array (Output block)

For an input block, the array field output port must be connected to the input of a Demux block with 256 outputs. All Demux block outputs will then have a dimension of 1. The output port just above (`rawDataCount` in the example) is used as a counter and indicates the number of items effectively valid, remaining items being empty. If the counter is greater than 256, only the first 256 items will be accessible.

For an output block, the array field input port must be connected to the output of a Mux block with 256 inputs. All Mux block inputs must have a dimension of 1. The input port just above (`rawDataCount` in the example) must also be connected and must be equal to the number of valid items in the array. It is not possible to send more than 256 items.

V.D.9.iii - Structures

Structured fields are generally put into variable size arrays.

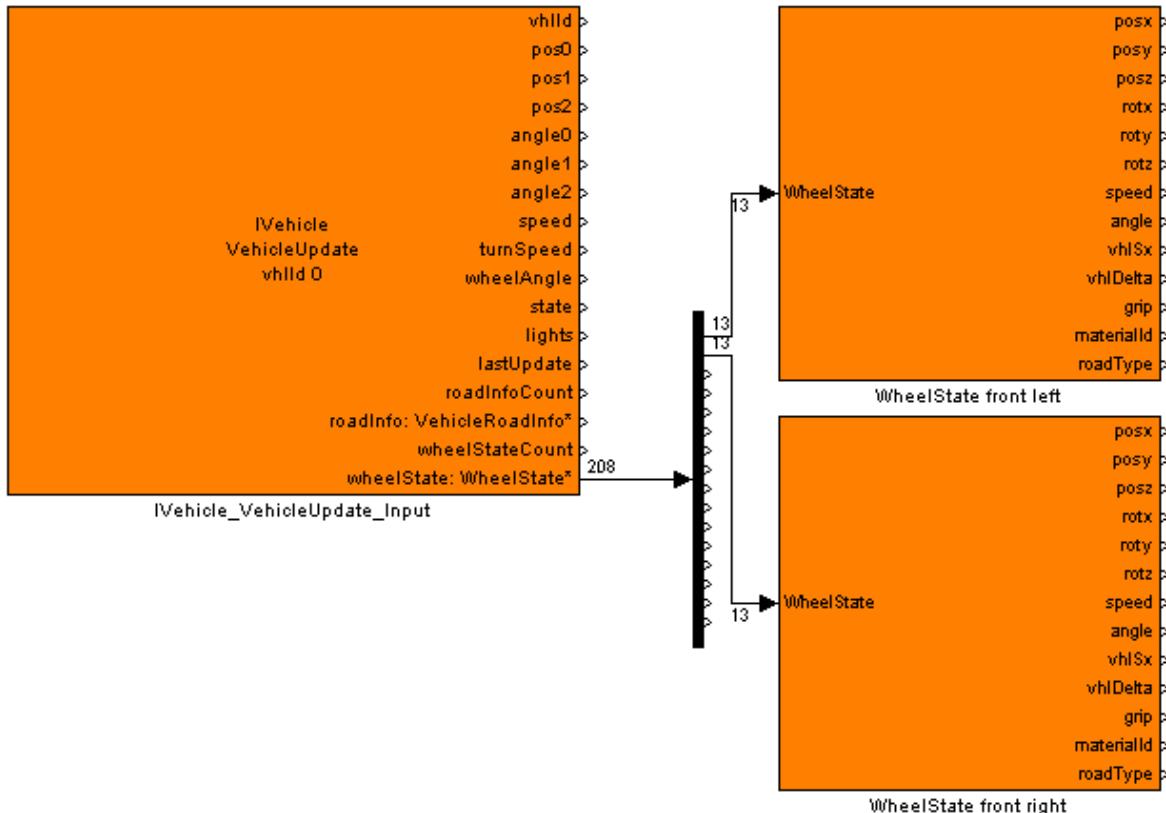


Illustration 47: Simulink – Structure (input block)

For an input block, the structured field output port must be connected to the input of a Simulink Demux block with 256 outputs. All Simulink Demux block outputs must then be connected to the input of a Demux block corresponding to the SCANeR structure. Displaying the signals dimensions allows checking the connection. The output port just above (`wheelStateCount` in the example) is used as a counter and indicates the number of items effectively valid, remaining items being empty. If the counter is greater than 256, only the first 256 items will be accessible.

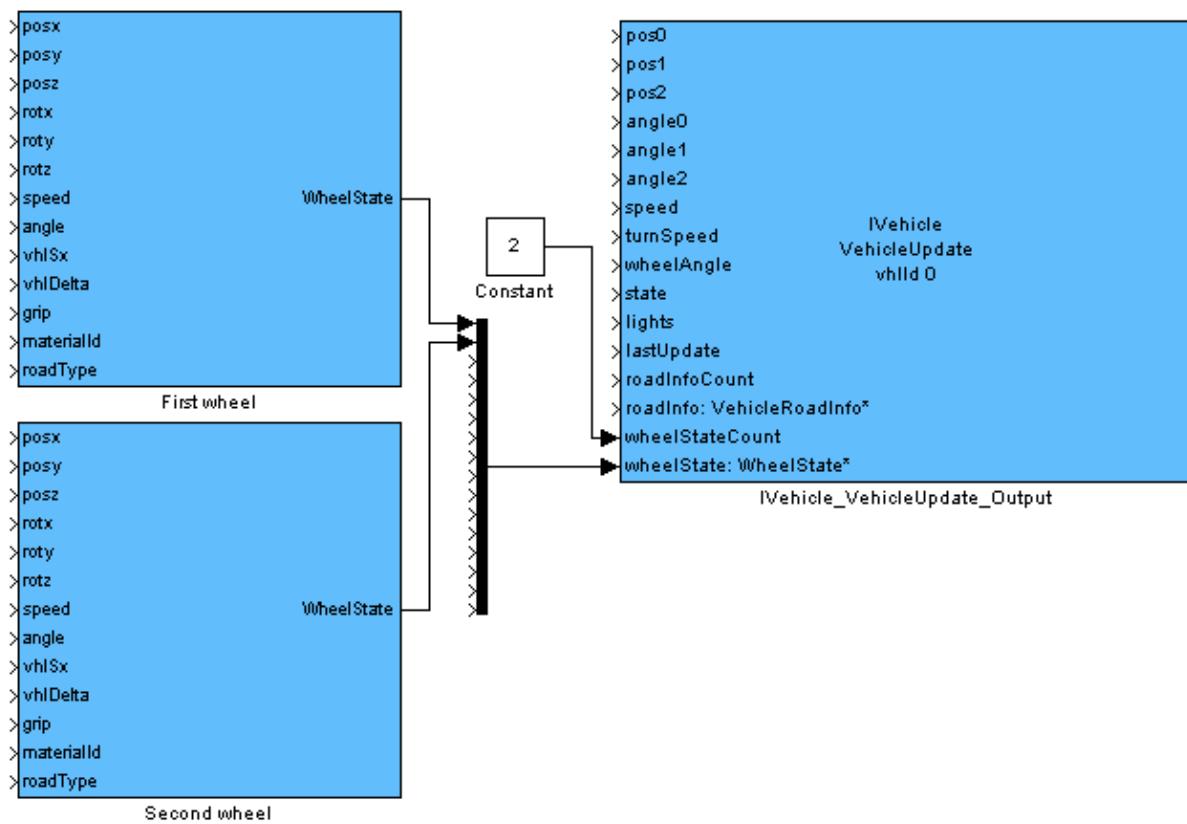


Illustration 48: Simulink – Structure (output block)

For an output block, you must create many Mux blocks corresponding to the SCANeR structure, and connect their outputs to the input of a Simulink Mux block with 256 inputs. The Simulink Mux block output must then be connected to the structured field input port. Displaying the signals dimensions allows checking the connection. The input port just above (`wheelStateCount` in the example) must also be connected and must be equal to the number of valid items in the array. It is not possible to send more than 16 items.

V.E - CO-SIMULATION

V.E.1 - SCANeR™ STUDIO CONFIGURATION

It is possible to use the Simulink model in a SCANeR simulation without compiling it. The Matlab application is then acting as a SCANeR module. To do that, it is necessary to add it in the process list of the current configuration. This process is not started from the studio GUI, so it is not necessary to give it an executable name:

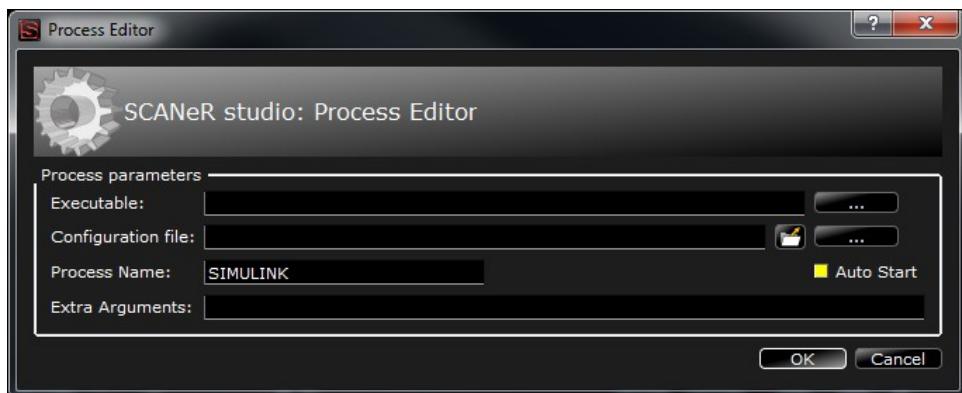


Illustration 49: Simulink – As a module in configuration

V.E.2 - SIMULINK MODEL CONFIGURATION



The communication between the model and SCANeR is managed by the Controller block. The Simulink model must contain one and only one Controller block.

The configuration name must be the one of the configuration currently used in SCANeR™ studio, and the process name must be the one chosen at the previous step:



Illustration 50: Simulink – Controller block

V.E.3 - STARTING AND STOPPING THE SIMULATION

To start the simulation, actions must be done in that order:

- Start the simulation in Simulink (CTRL+T)
- A Stop symbol (green square) should appear over the icon associated to the Simulink module in the SCANeR™ studio Simulator Status window
- Start the SCANeR™ studio simulation
- A Play symbol (green arrow) should appear over the icon associated to the Simulink module

To stop the simulation:

- Stop the SCANeR™ studio simulation
- A Stop symbol (green square) should appear over the icon associated to the Simulink module
- Stop the simulation in Simulink
- The Stop symbol should disappear from the icon associated to the Simulink module

V.E.4 - USING THE LAUNCHER BLOCK

The launcher block is used to control the simulation from Simulink. It can start the selected modules, load the scenario, play the simulation, and stop it.

- To start the simulation, start the simulation in simulink. The launcher will then load the scenario in SCANeR™ studio, start the “auto-start” modules, and play the simulation.
- To stop the simulation, click the stop button in Simulink. The launcher will then stop the simulation in SCANeR™ studio, close the scenario and stop the modules.

In every case, if SCANeR™ studio doesn't answer, the launcher will stop the simulation after a timeout which can be setted in the Launcher block parameters (default : 2 min).



If the scenario is already loaded, or the modules are already started, the simulation will directly start.



The Launcher block must have a higher priority than the Controller block. The block priorities can be set by clicking a block, selecting “Block Priority”, and setting a priority value. The lower the value, the higher the priority. For example, setting a priority of 0 for the Launcher block and 1 for the Controller block will allow the Launcher to start first.



If using the **OFFLINESCHEDULER** to play the simulation in non-realtime, **OFFLINESCHEDULER** autostart must be unchecked in the configuration GUI for SCANeR™ studio. Moreover, the IsSynchronized checkbox must be checked in the parameters window of the Launcher block.

V.E.5 - SEQUENCE DIAGRAM

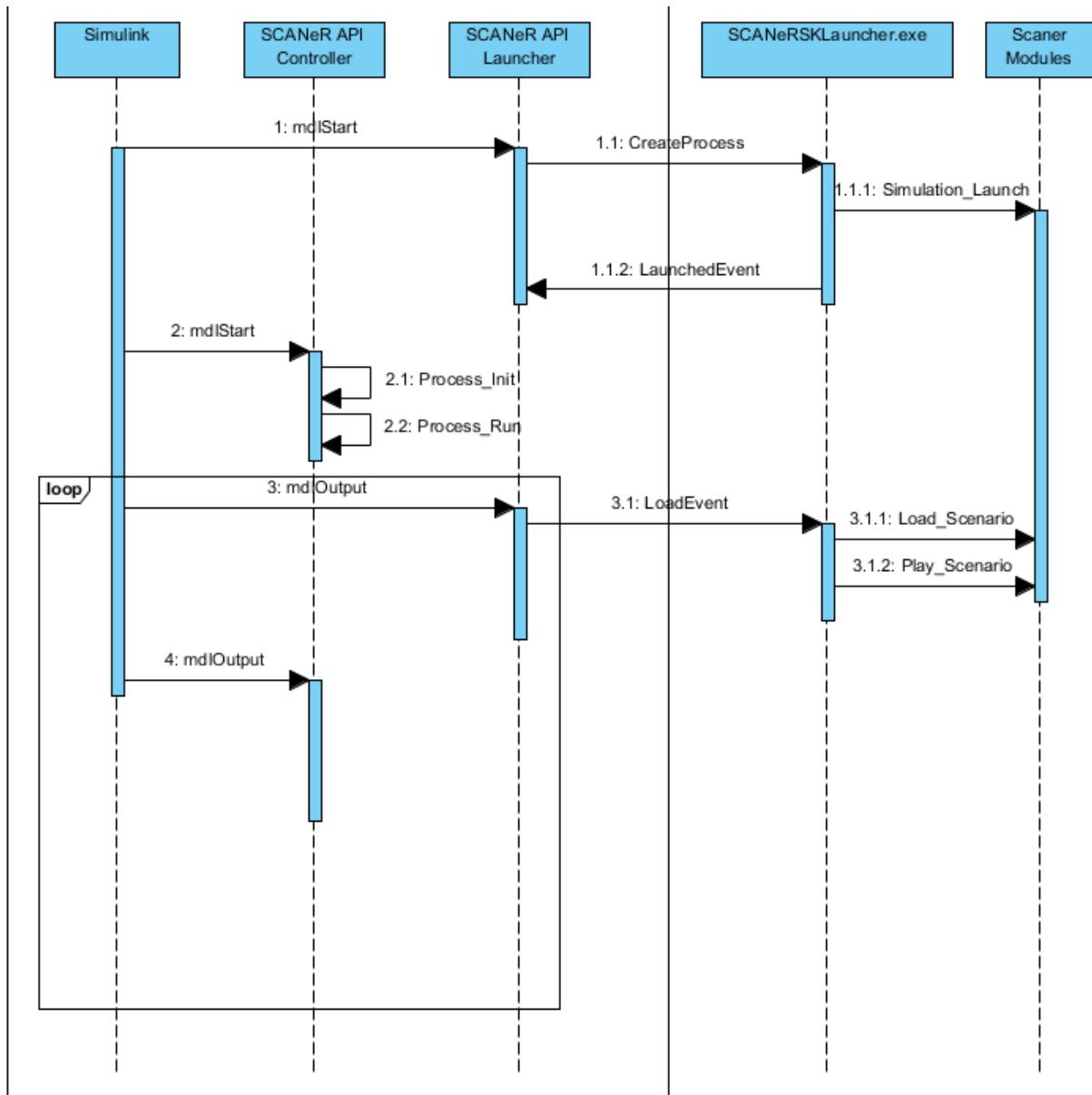


Illustration 51: Sequence Diagram for Cosimulation using the Launcher and Controller blocks

V.F - CODE GENERATION

V.F.1 - TARGET CHOICE

Two compilation targets are provided, `grt_scanerapi.tlc` and `ert_scanerapi.tlc`. They inherit respectively from `grt.tlc` and `ert.tlc`. The ERT target allows generating a more optimized code but needs an extra license.

V.F.2 - COMPILATION

The compilation can be done using the Tools / Real-Time Workshop / Build Model menu (or CTRL+B).

If the compilation succeeds, the Matlab console must print the following message:

```
### Successful completion of Real-Time Workshop build procedure for model
```

An executable with the name of the model is generated in the current directory. This executable has dependencies only on the SCANeR DLL. It can be run on a computer where Matlab is not installed.

The configuration name and the process name set in the Controller block are not used in the generated code. They are provided by SCANeR™ studio during execution. However, it is necessary to keep this block. If this block is not present in the model, the compilation will fail with the following error:

```
ert_scanerapi_main.obj : error LNK2019: unresolved external symbol _netOutFreqInCmdLine referenced in function _parseArgs
```

V.F.3 - COMPILATION WITHOUT STUDIO

It is not possible to do a co-simulation on a computer where SCANeR™ studio is not installed, but it is possible to generate code. To do that, do the 2 following setup actions:

- Install the Visual Studio runtime provided with SCANeR™ studio. It can be found in `<STUDIO_PATH>\<STUDIO_VERS>\third party\vcredist_x86.exe`.
- Copy the `<STUDIO_PATH>\<STUDIO_VERS>\APIs` directory on the computer.

V.F.4 - RUNNING

The generated executable is run the same way that other SCANeR modules. To be sure that the module finds its dependencies, it is advised to put it in the `<STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013` folder.

The module gets the configuration name and its process name from the command line. These arguments are automatically provided by SCANeR™ studio. It is also possible to give it 2 extra arguments:

- `-f FREQ`: model execution frequency (in Hz). The default value is the one set in the model Controller block.
- `-netfreq FREQ`: network messages emission frequency (in Hz). The default value is the one set in the model Controller block.

V.G - SAMPLES

Some samples are provided in that directory:

STUDIO_PATH>\<STUDIO_VERS>\APIs\ScanerAPI\Simulink\Samples.

If SCANEr™ studio has been installed in another directory, it is necessary to update the path to the libraries which is written in the sample files. To do that, you have to re-select the compilation target:

- Menu Simulation / Configuration Parameters (or CTRL+E)
- Category Real-Time Workshop
- Click on Browse
- Select ert.tlc
- Select ert_scanerapi.tlc and validate
- Category SCANEr
- Check that the indicated path is the one where SCANEr™ studio is installed
- Validate



Simulink file extension is “.mdl”. A “.mdl” contains information about system and subsystem. These files are not to be confused with vehicle files provided by OKTAL.

V.G.1 - APICONTROL

This sample shows how to launch a predefined scenario (DEFAULT\Studio_SCANEr_API_Simulink.sce) in a predefined configuration:

Prerequisites

- Create a process **SIMULINK** (without path) into your configuration
- Launch SCANEr™ studio with DEFAULT_1.5 configuration
- Run the APIControl.mdl example



By default the launcher timeout is set to 20s. Some SCANEr modules, as **PHYSICS**, may be longer than 20s to start the first time, so if needed set the default timeout. To do so, double click on the launcher block.



Only one SUPERVISOR can exist. If you run APIControl sample in Co-simulation, the SUPERVISOR must be declared on the same computer as Matlab.

V.G.2 - LANEKEEPING

This sample shows how to send vibrations on the steering wheel when the driver diverges from the lane center. The lane lateral shift is sent by the script. The model acts on the steering wheel and produces an additive torque if necessary.

Since this model uses the SHM, it must run on the same computer that **MODELHANDLER**.

Prerequisites

- Devices: a force feedback steering wheel
- Terrain: any
- Vehicle: an interactive vehicle with id 0
- Script: see below
- Modules: **SCENARIO**, **MODELHANDLER**, **ACQUISITION** and **SIMULINK**

Script

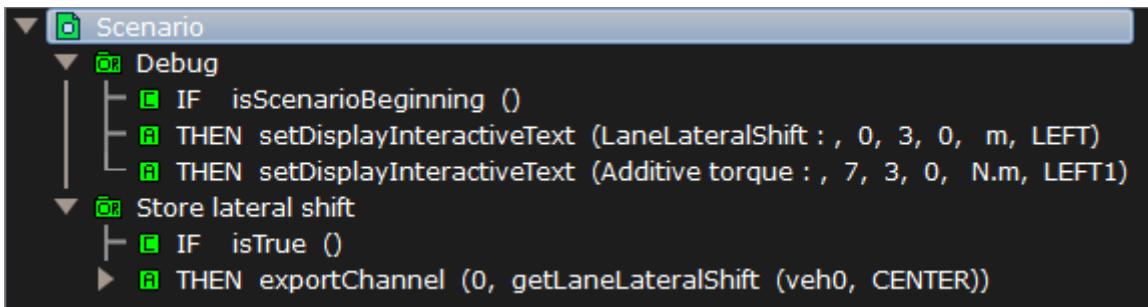


Illustration 52: Script

The controlled vehicle is the one with id 0. To control another vehicle, modify the **ModelCabin_SteeringToCabCorrective_Output** block in the "Send additive torque" subsystem.

The power and the duration of the vibrations are fixed in the model. To change them, modify the UpTorque, UpDuration, DownTorque and DownDuration blocks.

V.G.3 - RVLV

This sample shows how to implement a cruise control and a speed limiter. It acts on the throttle to control the vehicle speed.



To start the cruise control, the vehicle speed needs to be greater than 30 Km/h.



This model uses SHM, it must run on the same computer as **MODELHANDLER**.

Prerequisites

- Compile "RVLV.mdl" file with matlab/simulink.
- Terrain: Any
- Vehicle: A Callas vehicle with id 0
- Script: None
- Modules: **RVLV**, **MODELHANDLER**, **ACQUISITION** and **SIMULINK**

- Module **ACQUISITION**: Configure a HDRV file in order to use the first 5 "Custom" functions. (Note that by default OKTAL provides its "Keyboard.hdrv" already configured for this test. It uses F1 key for custom 0 function, F2 key for custom 1 function, F3 key for custom 2 function, F4 key for custom 3 function and F5 key for custom 4 function, see below "Function controls").

Function controls

- Custom 0: Turn off the cruise control.
- Custom 1: Turn on the cruise control (When it is activated the cruise control takes the current vehicle speed as initial value). To allows the vehicle to turn on the cruise control, the vehicle speed must be over 30 Km/h.
- Custom 2: Turn on the speed limiter (Acts on the throttle to limit the vehicle speed).
- Custom 3: Increase the vehicle speed of 1 Km/h.
- Custom 4: Decrease the vehicle speed of 1 Km/h.



A couple of data is exported in the exports channels 0 and 1 (0 shows the state mode, 1 shows the target speed).



In the subsystem the "Speed regulator" output is multiplied by the RegulMode ENABLE bloc to reset the subsystem during the mode changes. The Proportional–Integral–Derivative controller in this subsystem is also reset and does not accumulate mistake while it is not used.

V.G.4 - SPEEDLIMITOR

This sample shows how to implement a speed limiter. It acts on the throttle to limit the vehicle speed to 50 km/h.



This model uses SHM, it must run on the same computer as **MODELHANDLER**.

Prerequisites

- Terrain: Any
- Vehicle: A Callas vehicle with id 0
- Script: None
- Modules: **MODELHANDLER**, **ACQUISITION** and **SIMULINK**

The controlled vehicle is the one with id 0. To control another vehicle, modify the **ModelCabin_ModelToCab_Input** and **ModelCabin_CabToModelCorrective_Output** blocks.

The speed limit is fixed to 50 km/h. To change it, modify the Target block.



In the subsystem the "Speed limitor" output is multiplied by the LimitMode ENABLE bloc to reset the subsystem during the mode changes. The Proportional–Integral–Derivative controller in this subsystem is also reset and does not accumulate mistake while it is not used.

V.G.5 - SCANeR_API_DATAARRAY

To use this sample you need to:

- Create a scenario with a vehicle ID 0,
- Add to your SCANeR™ configuration one **SIMULINK** module for Co-simulation,
- Set up the Controller configuration name.

This sample illustrates how to read different type of values (refer to [Network.html](#) for additional details):

- 1 single value (here engineSpeed),
- 1 static array of standard type (here speed:float[6]),
- 1 dynamic array of structure type (here wheelState: WheelState*).

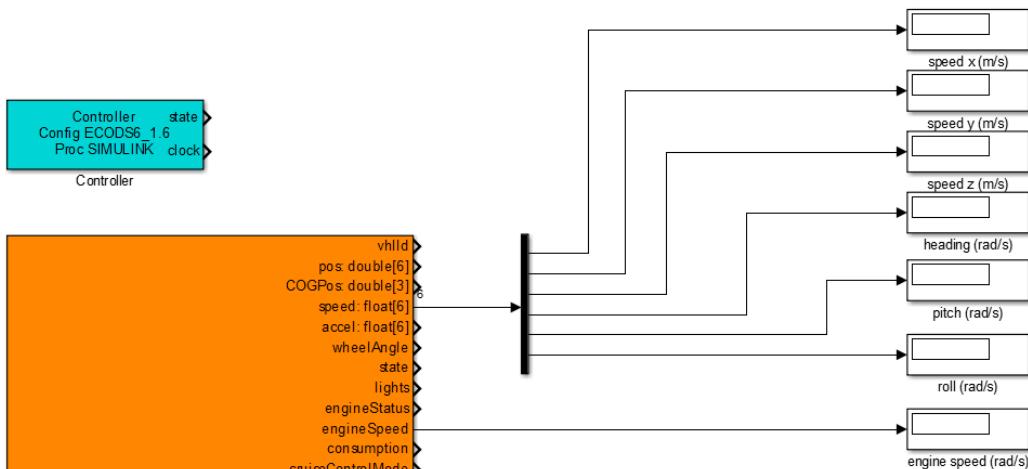


Illustration 53: read static array of standard data type

Dynamic array structure like “wheelState: WheelState*” need a 256 demux. You have to use the first ones you need. For our sample, 4 is corresponding to the 4 wheels of our car.

Note: If you need 8 wheels, then you will have to get the 8 first outputs of the demux.

Each wheel has its own WheelState Demux:

- 1st axle → Left
- 1st axle → Right
- 2nd axle → Left
- ...
- nth axle → Left

- nth axle → Right

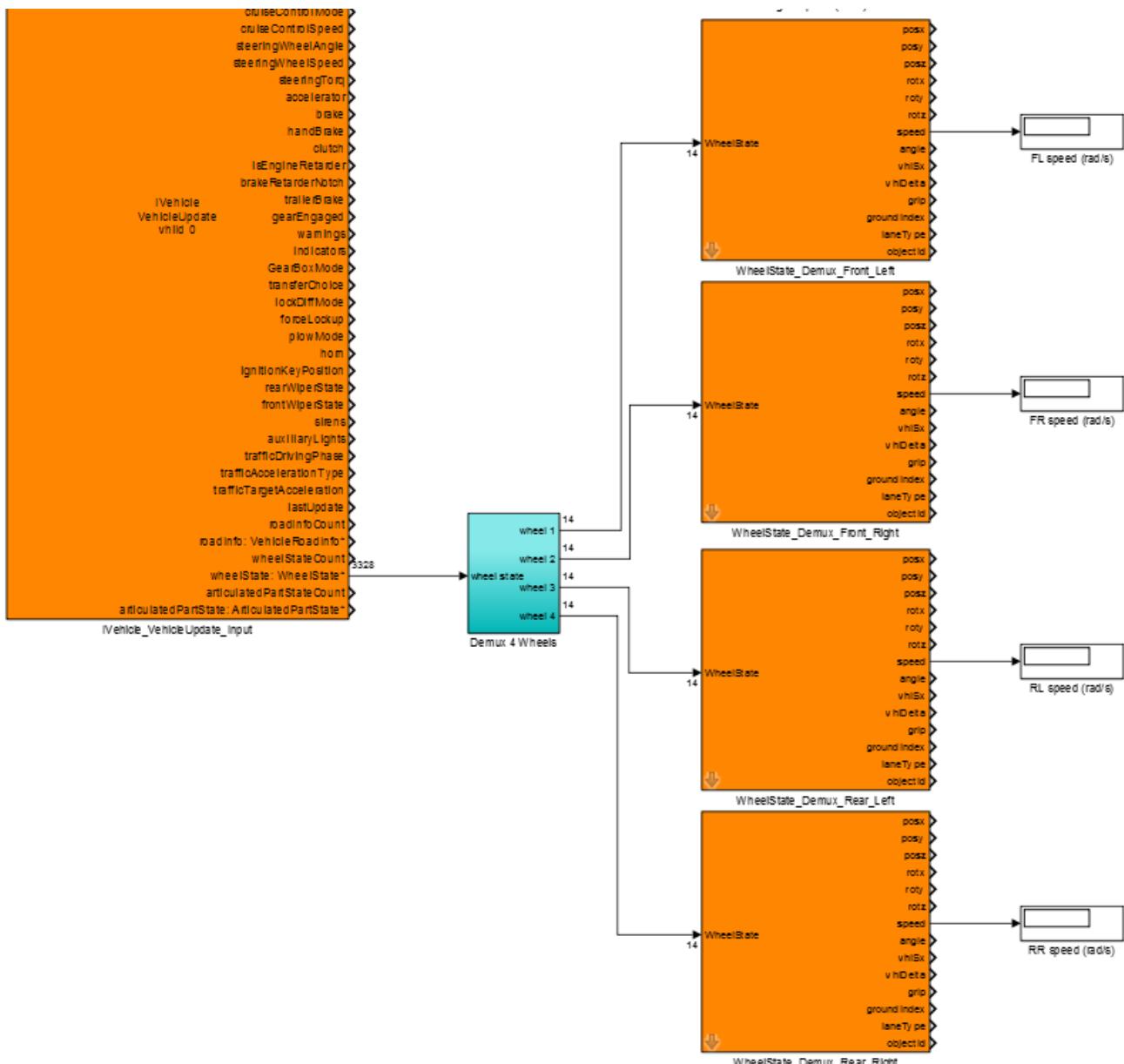


Illustration 54: read dynamic array of structure data type

VI - SCANNER API WITH RTMAPS

VI.A - INTRODUCTION

The SCANNER API RTMaps packages allow RTMaps and SCANNER™ studio softwares to communicate by Co-Simulation. The Co-Simulation allows you to have access to all the simulation data and to act on simulator components from an RTMaps diagram.

VI.B - FIRST STEPS

VI.B.1 - ENVIRONMENT SETUP

For details about environment setup refer to chapter “IV.B - Environment“.

One package is delivered per SCANNER™ interface (*interfaces are described into the Network.html and Shm.html documents*). Each interface allows you to send and receive messages.

VI.B.2 - INSTALLATION OF SCANNER API RTMAPS PACKAGES

To install the SCANNER API RTMaps packages follow instructions below:

1. Add to your PATH the directory containing **SCANNER_API_C_1.X.dll** (**<STUDIO_PATH>\<STUDIO_VERS>\bin\<ARCHITECTURE>\vs2013**),
2. Copy and paste the downloaded folder **Packages Studio 1.X** under **%RTMAPS_SDKDIR%\packages**,
3. Run RTMaps,
4. Register the packages which contain the messages you want to use.



The **SCANNER_Controller.pck** package is mandatory in all cases, it must be registered first.

VI.B.3 - RTMAPS PACKAGES

VI.B.3.i - ProcessManager

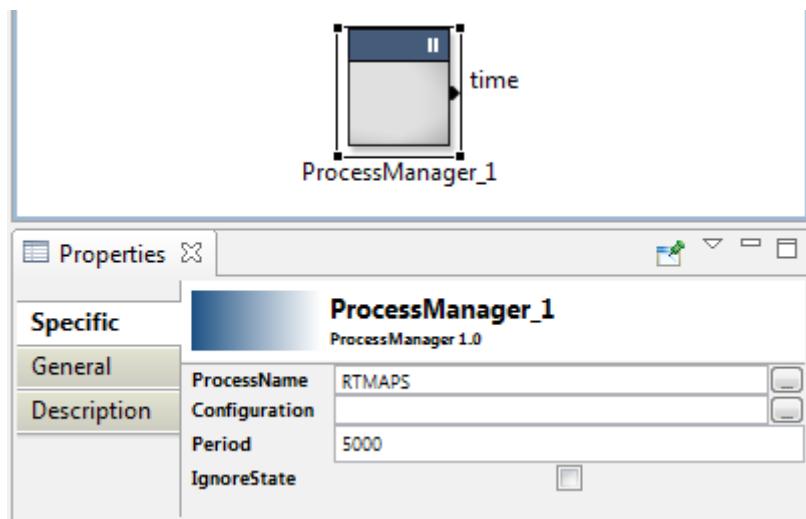


Illustration 55 : ProcessManager component

In **SCANEr_Controller.pck** package is found the ProcessManager component, which allows RTMaps to be detected as a SCANEr™ module (*simulation state management and scheduling*).

An RTMaps diagram must contains one and only one ProcessManager component.

Its properties are:

- ProcessName: Each module participating in the simulation must have a distinct process name. Type the process name assigned to RTMaps. This name must also be present in the SCANEr™ configuration.
- Configuration: Must be equal to the name of the current SCANEr™ configuration name. The current configuration name is displayed on the SCANEr™ studio title bar.
- Period: Execution frequency of the RTMaps diagram (*unit: microseconds*).
- IgnoreState: If checked the RTMaps diagram sends messages permanently without taking into account the SCANEr™ simulation state (*RUNNING, PAUSED, LOADED, etc.*).

VI.B.3.ii - Inputs / outputs packages

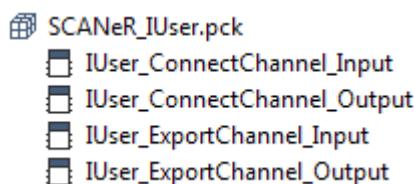


Illustration 56 : Inputs and outputs of packages

A package represents an interface and it contains the list of all its messages.

A message is an RTMaps component which gives access to its attributes.

An input component is used to retrieve simulation data in the RTMaps diagram.

An output component is used to send RTMaps diagram outputs to other SCANeR modules.

Message identification:

INTERFACE_MESSAGE_INPUT/OUTPUT

Some samples:

- **IUser_ExportChannel_Input** is to read data going through an export channel.
- **IUser_ExportChannel_Output** is to send data though an export channel.

The Lists of SCANeR™ studio Network messages and Shared memory structures are provided in [Network.html](#) and [Shm.html](#) documents.

VI.C - DIAGRAM AUTHORING

VI.C.1 - NETWORK AND SHM INPUT COMPONENTS

Input components allow using simulation data in the RTMaps diagram. These components have output ports but don't have input ports.

Their name is made up with the interface and the message name followed by "Input".

Each output port represents a field of the message. When a message is received, all fields are updated at the same time step. You just have to connect output ports to other RTMaps components to use the data in the diagram.

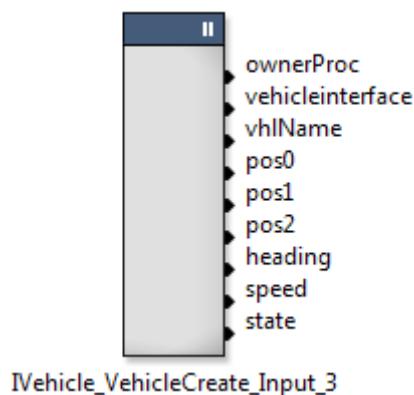


Illustration 57 : RTMaps Input component

VI.C.2 - NETWORK AND SHM OUTPUT COMPONENTS

Output components allow sending RTMaps diagram outputs to other SCANeR™ modules. These components have input ports but don't have output ports. Their name is made up with the interface and the message name followed by "Output".

Each input port represents a field of the message. A message is always sent with all of its fields. It is important to fill all fields in order not to send incoherent values to other modules. You just have to connect other RTMaps components to the input ports to send diagram data.

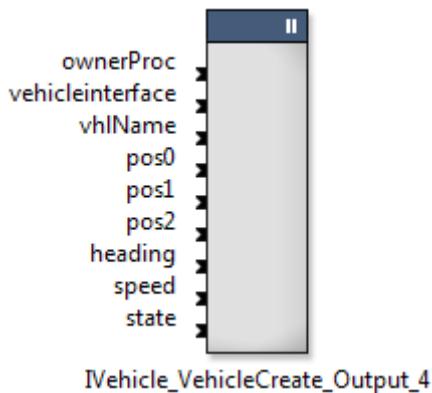


Illustration 58 : RTMaps Output component

VI.C.3 - SHM COMPONENTS

These components allow reading or writing in the shared memory. The shared memory is used by some modules to establish a high performance communication with **MODELHANDLER**. To read or write in the shared memory, RTMaps must run on the same computer than **MODELHANDLER**.



Currently only the vehicle id 0 can be managed by the SHM components

VI.C.4 - NETWORK COMPONENTS

These components allow exchanging network messages with other SCANeR™ modules. Since messages are sent using a multicast address, RTMaps can run on any computer participating in the simulation.

For input component, as long as no message has been received, all output ports have a zero value. After at least one message has been received, output ports have the value of the last received message. If the RTMaps diagram runs faster than the module sending the message, several model steps will be done with the same values for this component outputs. If the diagram runs slower, it will have a subsampling of the values.

For output components, a message is sent at the specified output frequency, even if the values computed by the model didn't change.

VI.C.5 - ARRAYS AND STRUCTURES

Most signals are of dimension 1, but some of them are arrays or structures.

VI.C.5.i - Fixed size arrays

For an input component, the number of output ports is related to the array size.

For an output component, the number of output ports is related to the array size.

VI.C.5.ii - Variable size arrays



Variable size arrays are not currently implemented.

VI.C.5.iii - Structure arrays



Only the first item of the array is returned.

VI.D - CO-SIMULATION

VI.D.1 - SCANeR™ STUDIO CONFIGURATION

To use an RTMaps diagram in a SCANeR simulation it is necessary to add into the process list of the current configuration an RTMaps process. This process is not started from the studio GUI, so it is not necessary to give it an executable name:

VI.D.2 - RTMAPS DIAGRAM CONFIGURATION



The communication between the diagram and SCANeR is managed by the ProcessManager component. The RTMaps diagram must contain one and only one ProcessManager component.

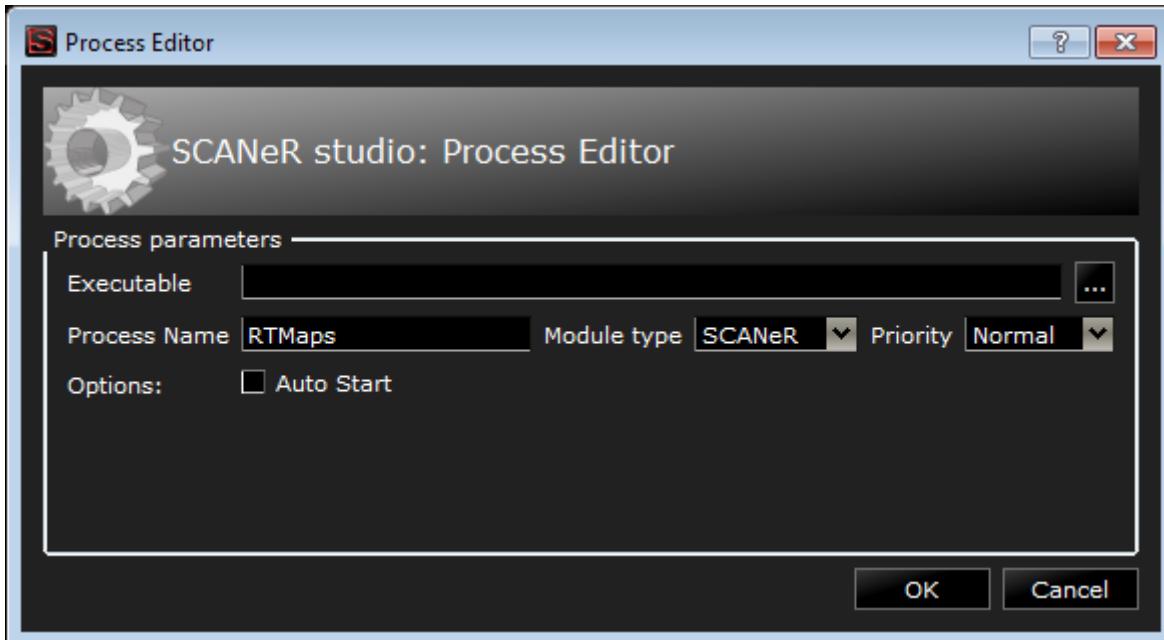


Illustration 59 : RTMaps module

The configuration name must be the one of the configuration currently used in SCANeR™ studio, and the process name must be the one chosen at the previous step:

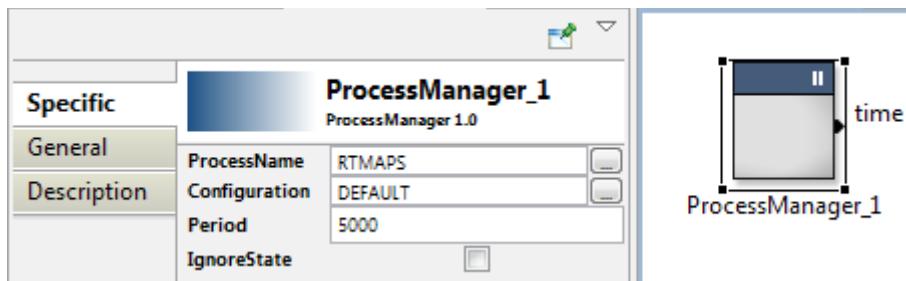


Illustration 60 : RTMaps Process Manager configuration

VI.D.3 - STARTING AND STOPPING THE SIMULATION

To start the simulation, actions must be done in that order:

1. Start the simulation in RTMaps (*a Stop symbol, green square, should appear over the icon associated to the RTMaps module in the SCANeR™ studio Simulator Status window*)
2. Start the SCANeR™ studio simulation (*a Play symbol, green arrow should appear over the icon associated to the RTMaps module*)

To stop the simulation:

1. Stop the SCANeR™ studio simulation (*a Stop symbol, green square, should appear over the icon associated to the RTMaps module*)
2. Stop the simulation in RTMaps (*the Stop symbol should disappear from the icon associated to the RTMaps module*)

VI.E - LIMITATIONS

- Only the vehicle id 0 can be managed by the SHM components
- Event system is not currently implemented.
- Variable size arrays are not currently implemented.
- Only the first item of a structure array is returned.

VII - ANALYSING TOOL REPLAY CONTROL API

VII.A - INTRODUCTION

Analysing Tool replay control plugin allows to add external actions when replaying a record file and manipulating the replay time (eg: pause, play, step forward etc..).

VII.B - ENVIRONMENT

Refer to the "Environment" chapter (page 18).

VII.C - PACKAGE

The Analysing Tool Replay Control API package includes:

- An include file, **ReplayControlPlugin.h** (located in the **<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\AnalysingToolAPI** folder).
- This documentation
- A MS Visual C++ 13 solution which contains a sample written in C++: **SampleReplayControlPlugin**
- To use an Analysing Tool plugin, the plugin file (dll) must be placed in the directory **<STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM⁸>\AnalysingToolPlugins** depending on the AnalysingTool binary platform.

VII.D - SAMPLES COMPILATION

Get the needed third party software according to your needs (see 111) or ask to OKTAL the corresponding package.

In the APIs/workspaces folder edit (with text editor) the APIs.vsprops file to update paths according to your environment (location of third parties software).

Open the complete.sln with compiler described in the "Environment" chapter (page 18) and compile needed samples. Execution of compiled samples can needed additional DLL from third party software so don't forget to copy needed DLL with your exe our add the DLL folder into your PATH environment variables.

⁸ The target is "win32" or "x64". It depends on the build version to use (32-bit or 64-bit version), in order to take advantage of 64-bit processors or in order to be compliant with 32-bit third-party software or device.

VII.E - REPLAY CONTROL PLUGIN INTERFACE

VII.E.1 - METHODS

Function name	Function description
<code>virtual const char* name() const = 0;</code>	Return the name of the plugin.

Function name	Function description
<code>virtual const char* version() const = 0;</code>	Return the plugin version.

Function name	Function description
<code>virtual const char* description() const = 0;</code>	Return the plugin description.

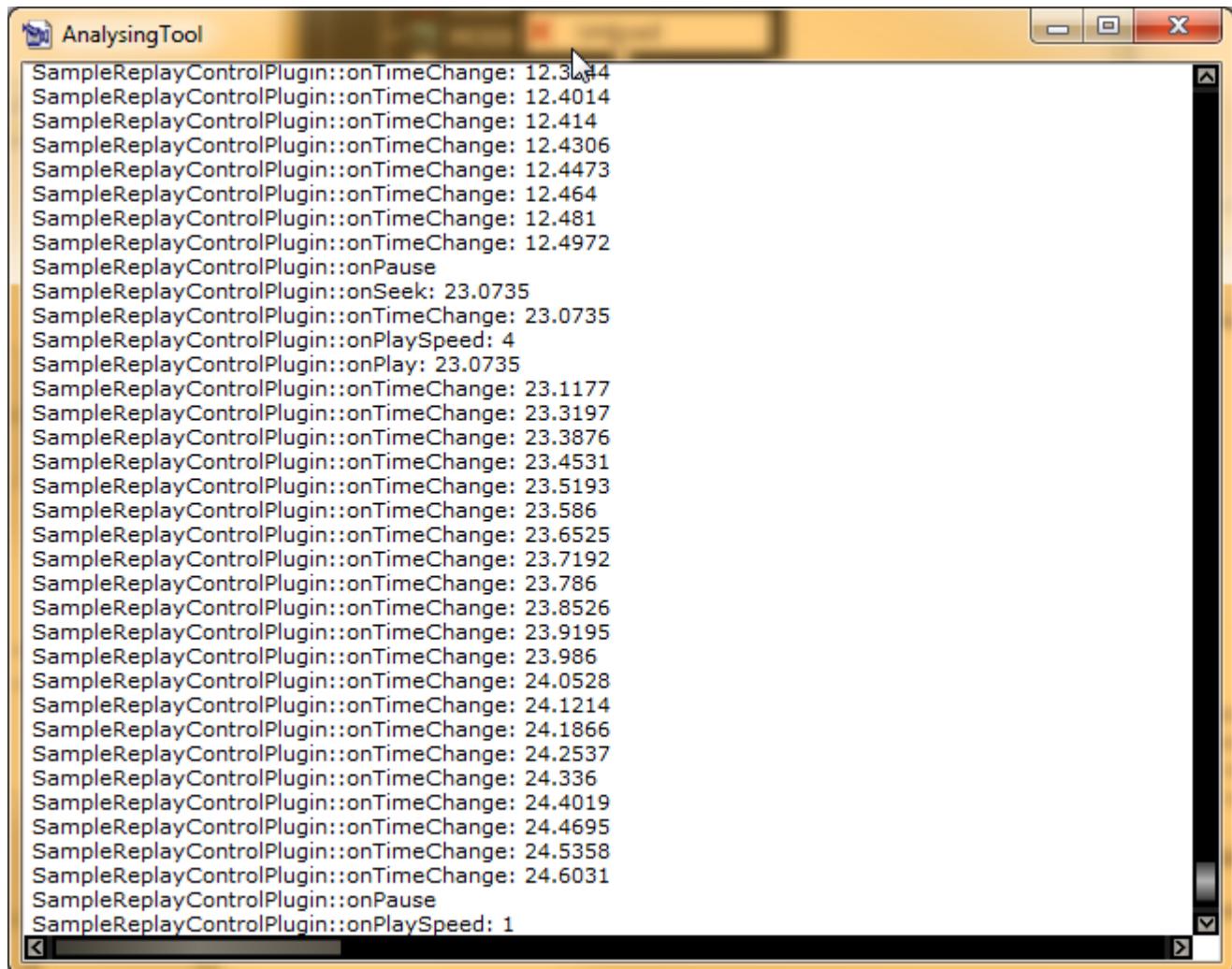
Function name	Function description
<code>virtual void onLoad() const = 0;</code>	Called when loading the plugin.

Function name	Function description
<code>virtual void onTick() const = 0;</code>	Called on each tick, even if the time does not change.
<code>virtual void onUnload() const = 0;</code>	Called when unloading the plugin.
<code>virtual void onPlay(double t) const = 0;</code>	Called on play. Param 1 - t: Start time in seconds
<code>Virtual void onPlaySpeed(double speedFactor) const = 0;</code>	Called when the playback speed is changed.
<code>Virtual void onPause() const = 0;</code>	Called on replay pause.
<code>Virtual void onSeek(double t) const = 0;</code>	Called when the user jumps to a specific time. Param 1 - t: Time in seconds
<code>Virtual void onStep(int step) const = 0;</code>	Called when the user steps forward or backward, Param 1 - step: is the step value and can be < 0
<code>Virtual void onTimeChange(double t) const = 0;</code>	Called only when the time change. Param 1 - t: Time in seconds

VII.F - DELIVERED SAMPLES DESCRIPTION

VII.F.1 - SAMPLEREPLAYCONTROLPLUGIN

The SampleReplayControlPlugin implement each method of the interface and just display a message in AnalysingTool logs (to display Analysing Tool logs select menu **HELP \ Analysing tool logs**).

A screenshot of a Windows application window titled "AnalysingTool". The window contains a single text log entry from the SampleReplayControlPlugin. The log entry is a long list of timestamped messages, all starting with "SampleReplayControlPlugin::onTimeChange:" followed by a timestamp. The timestamps range from 12.34014 to 24.6031, with some intermediate values like 12.414 and 23.1177. There are also two other types of messages: "SampleReplayControlPlugin::onPause" at timestamp 23.0735 and "SampleReplayControlPlugin::onPlaySpeed: 1" at timestamp 24.6031.

```
SampleReplayControlPlugin::onTimeChange: 12.34014
SampleReplayControlPlugin::onTimeChange: 12.4014
SampleReplayControlPlugin::onTimeChange: 12.414
SampleReplayControlPlugin::onTimeChange: 12.4306
SampleReplayControlPlugin::onTimeChange: 12.4473
SampleReplayControlPlugin::onTimeChange: 12.464
SampleReplayControlPlugin::onTimeChange: 12.481
SampleReplayControlPlugin::onTimeChange: 12.4972
SampleReplayControlPlugin::onPause
SampleReplayControlPlugin::onSeek: 23.0735
SampleReplayControlPlugin::onTimeChange: 23.0735
SampleReplayControlPlugin::onPlaySpeed: 4
SampleReplayControlPlugin::onPlay: 23.0735
SampleReplayControlPlugin::onTimeChange: 23.1177
SampleReplayControlPlugin::onTimeChange: 23.3197
SampleReplayControlPlugin::onTimeChange: 23.3876
SampleReplayControlPlugin::onTimeChange: 23.4531
SampleReplayControlPlugin::onTimeChange: 23.5193
SampleReplayControlPlugin::onTimeChange: 23.586
SampleReplayControlPlugin::onTimeChange: 23.6525
SampleReplayControlPlugin::onTimeChange: 23.7192
SampleReplayControlPlugin::onTimeChange: 23.786
SampleReplayControlPlugin::onTimeChange: 23.8526
SampleReplayControlPlugin::onTimeChange: 23.9195
SampleReplayControlPlugin::onTimeChange: 23.986
SampleReplayControlPlugin::onTimeChange: 24.0528
SampleReplayControlPlugin::onTimeChange: 24.1214
SampleReplayControlPlugin::onTimeChange: 24.1866
SampleReplayControlPlugin::onTimeChange: 24.2537
SampleReplayControlPlugin::onTimeChange: 24.336
SampleReplayControlPlugin::onTimeChange: 24.4019
SampleReplayControlPlugin::onTimeChange: 24.4695
SampleReplayControlPlugin::onTimeChange: 24.5358
SampleReplayControlPlugin::onTimeChange: 24.6031
SampleReplayControlPlugin::onPause
SampleReplayControlPlugin::onPlaySpeed: 1
```

Illustration 61 : SampleReplayControlPlugin logs

viii - SCENARIO API

viii.a - PRESENTATION OF API

viii.a.1 - INTRODUCTION

Scenario API provides an easy way to quickly modify a scenario without using SCANeR™ studio UI. This API is based on XPATH technology, that permit to modify each node of a given XML file.

For example this allow modification on the fly of the same scenario for recursive tests.

The dynamic Scenario API is delivered with a common way to use it.

To be compliant with a lot of application, the Scenario_API is delivered in the form of a DLL (which contains all needed methods), with a "C language" binding. It can be mapped with applications written in C, C++, C# or any application able to use a DLL.

In an easy way and without high programming background requirements, the following actions can be done:

- open a scenario existing in a given SCANeR™ configuration
- modify a given value of a scenario using XPATH.
- save the modified scenario

viii.a.2 - ENVIRONMENT

Refer to the "Environment" chapter (page 18).

viii.a.3 - PACKAGE

The SCANeR™ studio API package includes:

- A DLL file: **Scenario_API.dll** (located in the <STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013 folder) & .
- A LIB file: **Scenario_API.lib** (located in the <STUDIO_PATH>\<STUDIO_VERS>\APIs\lib\<PLATFORM>\vs2013 folder)
- An include file, **scenarioAPI_DLL_C.h** (located in the <STUDIO_PATH>\<STUDIO_VERS>\APIs\include\ScenarioAPI folder).
- This documentation
- A MS Visual C++ 8 solution which contains some examples written in C++
- A python sample (located in the <STUDIO_PATH>\<STUDIO_VERS>\APIs\samples\ScenarioAPI\python folder).



Warning: Next to a SCANeR™ studio release update (or a version update) you must to rebuild all your own modules, binaries and developments.

VIII.A.4 - SAMPLES COMPILATION

Get the needed third party software according to your needs (see 111) or ask to OKTAL the corresponding package.

In the APIs/workspaces folder edit (with text editor) the APIs.vsprops file to update paths according to your environment (location of third parties software).

Open the complete.sln with compiler described in the "Environment" chapter (page 18) and compile needed samples. Execution of compiled samples can needed additional DLL from third party software so don't forget to copy needed DLL with your exe our add the DLL folder into your PATH environment variables.

VIII.A.5 - INTERFACE OF THE SCENARIO API STUDIO API

VIII.A.5.i - Scenario initialisation

The initialisation of the API is done using the method: `int Scenario_Init(const char* configName);`

The parameter correspond to the configuration used by the API to find a scenario.

This method will return 0 if initialisation was done properly, and return 1 if an error occurred.

VIII.A.5.ii - Quit scenario API

To quit Scenario API, use the method: `void Scenario_Close();` at the end of your application.

VIII.A.5.iii - Load a scenario

A scenario is loaded using: `int Scenario_Load(const char* fileName);`

The parameter correspond to the name of the scenario to load, without extension. If the scenario is not found in the configuration given at the initialisation of the API or an other error occurred.

This method will return 0, and return 1 if the scenario was correctly loaded.

VIII.A.5.iv - Save a scenario

Use the method “`int Scenario_Save(const char* fileName);`” to save the previously loaded scenario in a file named “fileName”. The given name must be without extension, it will be automatically added during saving. The file is saved in the same directory of the previously loaded scenario.

This method returns 0 if an error has occurred during saving, and 1 if saved without problem.

VIII.A.5.v - Modify an element of a scenario

A scenario file is actually an XML file. This allow the use of technology that ease the search of elements in this kind of file, such as XPATH (a simple query language) .

The method “`int Scenario_ModifyNodeValue(const char* xPathRequest, const char* newNodeValue);`” is based on this technology.

The first parameter is an XPATH (a query that allow the API to retrieve the node to modify), and the second one is the value that will be set to this node.

XPATH is a technology normalised by the W3C, you can find more details about it at this page: <http://www.w3.org/TR/xpath>

VIII.A.5.vi - Example

In this example, you have a part a scenario file containing a vehicle and its initial speed. The following code is a C++ sample code using Scenario API which modify the initial speed of the car.

Part of the scenario file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<sce version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<Scenario>
...
<Vehicle>
<id>0</id>
...
<initialSpeed>0</initialSpeed>
...
</Vehicle>
...
</Scenario>
</sce>
```

C++ sample code:

```
// Initialisation of the API
Scenario_InitParam("ConfigurationName");

// Do nothing if an error has occurred during load
if (Scenario_load("Studio_trafficStopSign"))
{
    // Modify the initial speed of a car
    bool rqRes = Scenario_modifyNodeValue("/sce/Scenario/Vehicle[id=3]/initialSpeed", "60");

    // Display the request result:
    //           1 if the value was correctly changed
    //           0 if an error has occurred)
    std::cout << "Request Initial Speed set to 60 : " << rqRes << std::endl;

    // Save the modified scenario in a new file
    Scenario_save("Studio_trafficStopSign_copy");
}

// Quit scenario API
Scenario_close();
```

VIII.B - USE IN SCANeR STUDIO

There is a functionality in SCANeR™ studio that help the generation of an XPATH request from a scenario.

This option is available in the main menu View/Docks/SCENARIO TREE. Check the box to enable the scenario tree option.

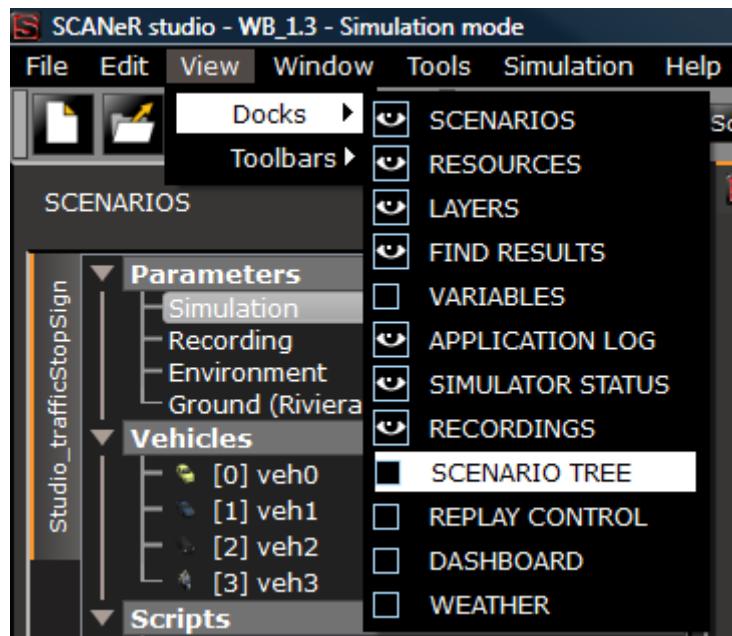


Illustration 62: Scenario tree option

When the option is activated, the XML tree of the current scenario is displayed on the “dock” SCENARIO TREE.

SCENARIO TREE	
Variable	Value
▼ sce	
▼ Scenario	
name	
description	Run TRAFFIC This scenario shows the capability o...
online	1
steadystate	0
► Environment	
► Aimsun	
► Visual	
► Ground	
► ProgressDlg	
► Recording	
► ScenarioScript	
▼ Vehicle(veh0)	
id	0
driverId	0
motionId	-1
name	veh0
modelName	Citroen_C3_Green
process	TRAFFIC
startType	0
initialDistOnTraj...	0
initialSpeed	0
state	0
trailerId	-1
initEngineRunning	1
recomputeRestit...	1
► ObjectPosition	
► ItineraryRoadXml	
► StopCriteria	
► Model	

Illustration 63: Scenario tree

When a node is selected, the corresponding XPATH is displayed at the bottom of the dock. To retrieve the XPATH of the selected element, right click on it and select between the three available options: XPATH of current node, value of current node, XPATH + value of current node.

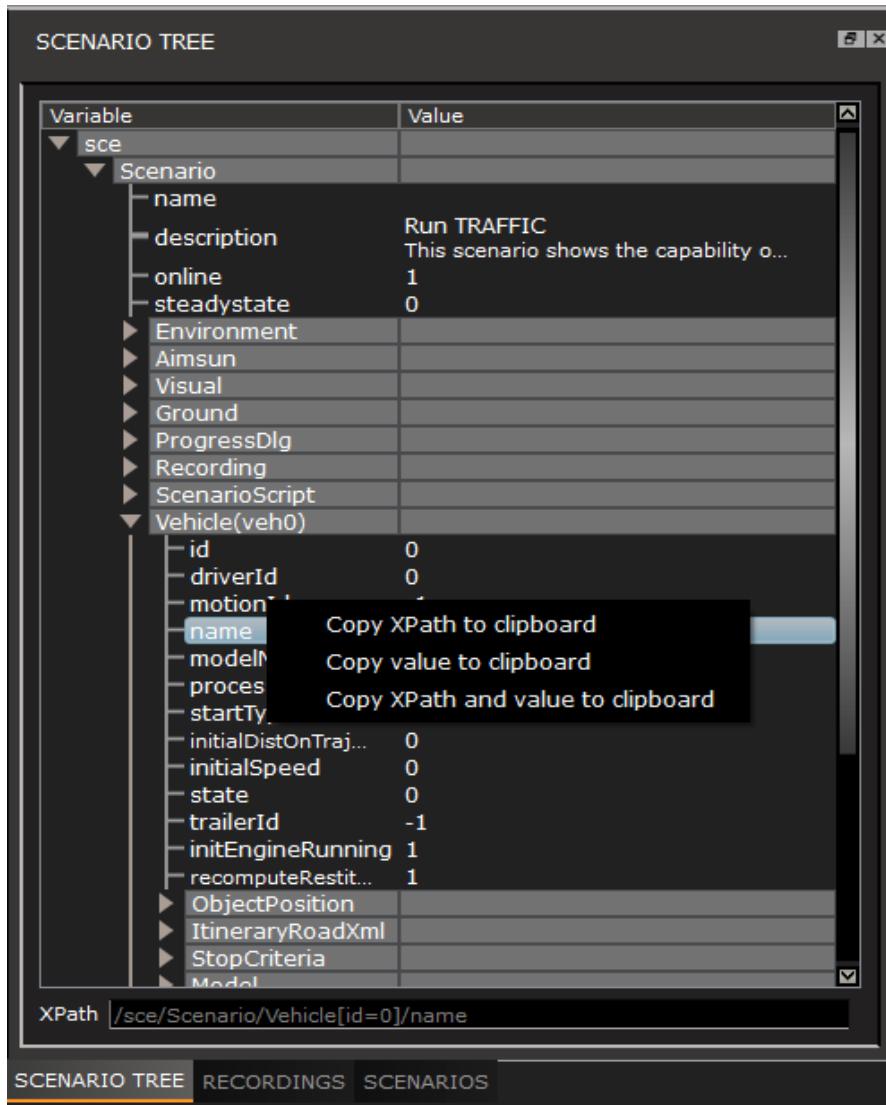


Illustration 64: Right click on the scenario tree

Here is the result of each option available on the right click menu, using the scenario displayed in Illustration II.3:

- Copy Xpath to clipboard: /sce/Scenario/Vehicle[id=0]/name
- Copy value to clipboard: veh0
- Copy Xpath and value to clipboard: "/sce/Scenario/Vehicle[id=0]/name", "veh0"

Note that the right click menu is available only for leaf nodes of the XML tree.

Note : If the Scenario is changed, the “dock” SCENARIO TREE will be updated after a save (Press “Scenario save” button).

VIII.C - SCENARIO SPECIAL ELEMENTS

The aim of this section is to shortly list interesting elements in a scenario.

VIII.C.1 - VEHICLE

VIII.C.1.i.a - Vehicle unique identifier

A vehicle is identified by an unique identifier in a scenario. This information is defined in the element “id” of a Vehicle element.

It is highly recommended to use this identifier when using the scenario API is to retrieve a vehicle in a scenario.

For example, to change the model name of the vehicle which id is 0 (Illustration II.2), the following XPATH can be used: “/sce/Scenario/Vehicle[id=0]/modelName”.

VIII.C.1.i.b - Vehicle position

The position of a vehicle is defined on the section “ObjectPosition” of the vehicle element.

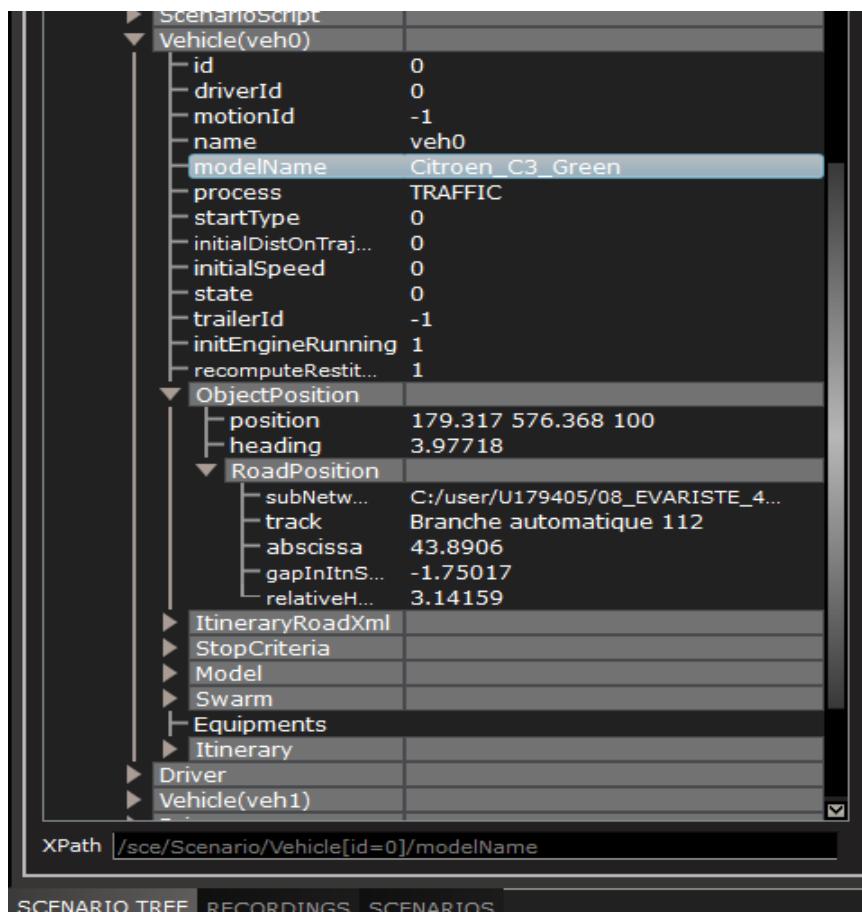


Illustration 65: Vehicle position

The section “Object position” contain two kind of description of position:

- an absolute one: defined by **position** and **heading**. This two information give the absolute position of the vehicle on the whole scene.
- A relative position: defined by the element **RoadPosition**. This element define the position of a vehicle relatively to the road in which the vehicle is.

To modify the relative position (abscissa on the track for example), this kind of Xpath can be used: "/sce/Scenario/Vehicle[id=0]/ObjectPosition/RoadPosition/abscissa".

VIII.C.2 - DRIVER

As the vehicle element, the driver is identified by an unique identifier in the scenario in the element named "id".

It is highly recommended to use this identifier to retrieve a Driver element in a scenario.

Here is an example of Xpath to modify the type of the driver which id is 0:

```
"/sce/Scenario/Driver[id=0]/driverType", "TrafficDriver"
```

VIII.C.3 - SCENARIO SCRIPT

A scenario Script is defined by an unique identifier in the element “scriptName”.

It is highly recommended to use this identifier to retrieve a Script element in a scenario.

Here is an example of Xpath to retrieve a Script from a given scenario:

“/sce/Scenario/ScenarioScript[scriptName="Main"]/scriptName”

SCENARIO TREE	
Variable	Value
sce	
Scenario	
name	
description	
online	1
steadystate	0
Environment	
Aimsun	
Visual	
Ground	
ProgressDlg	
Recording	
PredefinedRun	
ScenarioScript	
scriptName	Main
Variable	
name	0
label	SimulationDuration
value	33.000000
Vehicle(Citroen_C3)	
Driver	

XPath : scenario/ScenarioScript[scriptName="Main"]/Variable[name=0]/value

SCENARIO TREE RECORDINGS SCENARIOS

Illustration 66: Scenario Script element

VIII.C.4 - VARIABLES OF A SCENARIO SCRIPT

A scenario script variable is identified by an unique identifier in the scenario script, in the element named “name”.

It is highly recommended to use this identifier to retrieve a variable element in a scenario script.

Here is an example of Xpath to modify the value of the variable which name is 0 (Illustration II.C.1):

“/sce/Scenario/ScenarioScript[scriptName="Main"]/Variable[name=0]/value", "33.000000”

IX - HTTP API

This chapter describes how to use the HyperText Transfer Protocol to remotely access to some of the supervisor functions (Simulation mode, refer to SCANeRstudio_SIMULATION_UserManual documentation) and to some of the AnalysingTool functions (refer to SCANeRstudio_ANALYSIS_UserManual documentation).

IX.A - INTRODUCTION

IX.A.1 - TERMINOLOGY

Acronym	Signification
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
URI	Uniform Resource Identifier
API	Application Programming Interface
GUI	Graphical user interface
UTF-8	Universal Character Set Transformation Format - 8 bits

Table 14: HTTP API - Acronym

IX.A.2 - PRESENTATION

The API presented here is an HTTP API to control the SCANeR studio GUI. This API allows the user to build its own supervision application that fulfills the previous requirement.

This approach has the advantage to keep the existing advanced controls (like 3D supervision).

You will find in the "Samples" chapter (IX.B page 139) two basic remote controllers:

- Simple Studio controller HTML5 sample
- Simple Analysis controller HTML5 sample

IX.A.3 - OVERVIEW

HTTP API is a client/server application:

- User supervisor is the client
- SCANeR studio GUI is the server.

The client has to send HTTP requests to control the simulation, and SCANeR executes the requests.

It is up to SCANeR to manage directly the module states.

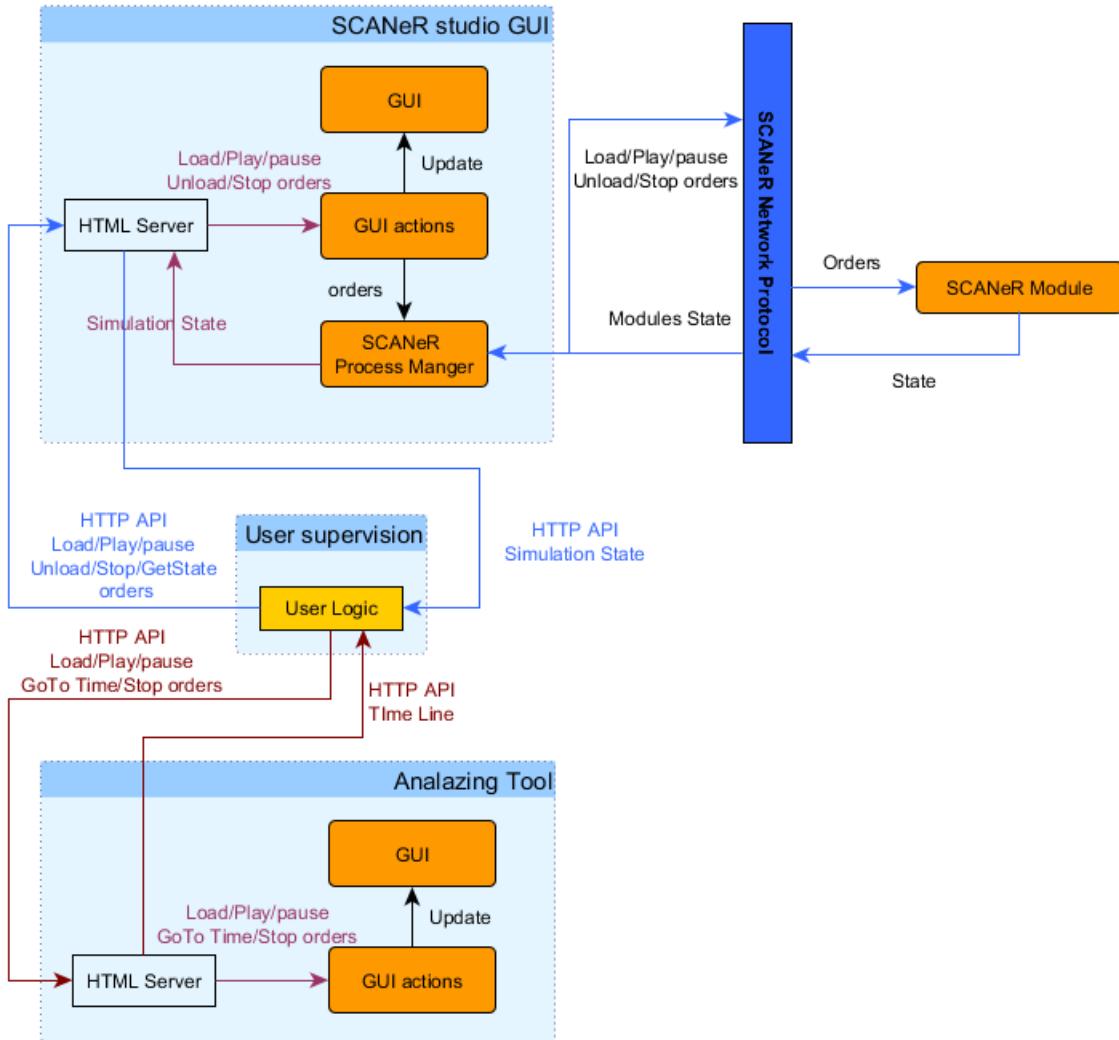


Illustration 67 : HTTP API - Http control embedded in studio GUI

This API is compatible with any OS and language.

It can be used by several clients: it is possible to have another client to supervise SCANeR with a touch pad.

The HTTP API does not require a SCANeR license, or any kind of SCANeR resource, to be used.

The HTTP API offers the possibility to be connected to more than one SCANeR simulation.

IX.A.4 - API DESCRIPTION

The API is accessible through the HTTP/1.0 or HTTP/1.1 protocol.

It is a set of URI to send the SCANeR studio GUI.

GET and POST methods are both supported to send the URI.

All strings are encoded in UTF-8 and are compatible with Japanese characters (just like SCANeR studio).

Usually the return message is immediate, and the long operations are done asynchronously.

When the URI succeed, the codes 200 or 204 are returned.

If the URI failed, the return HTTP code is 404.

IX.A.4.i - Studio HTTP API

Studio API is reachable through the port 9090 (the sample is described page 139).

URI	Description	HTTP return code
/api/scenarios	Returns the scenario list of the current configuration. It is returned as a JSON data string ⁹ .	200
/api/open?scenario=name	Opens the scenario named "name". This scenario becomes the current scenario. The return is immediate; there is no wait for the server to load the scenario. If the scenario is already opened it will not open twice, but it will become the current scenario.	204
/api/open?record=name	Starts AnalysingTool with the record named "name".	204
/api/play?scenario=name	Plays the scenario named "name". The scenario has to be opened first. If no "scenario=name" is given, the currently opened scenario is launched. If a record is loaded and paused, the play will resume the record. The return is immediate, there is no wait for the server to play the scenario.	204
/api/play?record=name	Opens and launches the record named "name". The return is immediate; there is no wait for the server to load the record.	204
/api/pause	Pauses the currently played scenario/record. The return is immediate; there is no wait for the server to pause the scenario.	204
/api/stop	Stops the currently played scenario/record. The return is immediate; there is no wait for the server to stop the scenario.	204
/api/state	Returns the module state and the GUI state as a JSON data string ⁹ .	200
/api/start	Starts all the modules with the "selected" property to ON. The return is immediate; there is no wait for the server to start the modules.	204
/api/kill	Kills all the modules. The return is immediate; there is no wait for the server to kill the modules.	204
/api/configurations	Returns the configuration list as a JSON data string ⁹ .	200
/api/configuration?name=config_name	Changes the current configuration to "config_name". The return is immediate; there is no wait for the server to change the configuration.	204
/api/path?name=data	Returns the data path as a JSON data string ⁹ .	200
/api/records	Returns the record list of the current configuration. Only records with GZ file are returned. It is returned as a JSON data string ⁹ .	200

Tableau 15: HTTP API - Studio

⁹ JSON data strings are described in the next chapter IX.A.5 - "JSON" - page 134.

IX.A.4.ii - AnalyzingTool HTTP API

AnalysingTool API is reachable through the 9091 port (the sample is described page 143).



Be careful this API is usable only if the AnalysingTool program is launched. It can be launched remotely using the `api/open?record=name` of the Studio HTTP API (page 133).

URI	Description	HTTP return code
<code>/api/pause</code>	Pauses the currently played record.	204
<code>/api/stop</code>	Same as pause.	204
<code>/api/play</code>	Play (records must be opened first).	204
<code>/api/open?record=name</code>	Opens the record named “name”.	204
<code>/api/close?record=name</code>	Closes the record named “name”.	204
<code>/api/time</code>	Returns the time in a JSON data string ⁹ .	200
<code>/api/time?time=time</code>	Go to time “time” for the current opened record.	204
<code>/api/quit</code>	Close AnalysingTool. Once this URI is called, no other api call can be used.	204

Tableau 16: HTTP API - AnalysingTool

IX.A.5 - JSON

JSON is a data interchanged format, simple, widely used and supported.

Its specification is available here [ECMA-404 The JSON Data Interchange Standard](#).

This section describes the different JSON data strings used in this API.

IX.A.5.i - api/scenarios

```
{“scenarios”:[
  “Scenario name1”,
  “Scenario name2”,
  ...
  “Scenario nameN”
]}
```

Table 17: HTTP API – JSON api/scenarios

The JSON data string is composed of 1 array named “scenarios”. This array is filled with all the SCANeR scenarios found in the current configuration. Each scenario is represented as a string filled with the scenario name.

IX.A.5.ii - api/state

```
{
  "states": [
    {"STUDIO": ["state name" ...]},
    {"module name 1": "state name"},
    {"module name 2": "state name"},
    ...
    {"module name N": "state name"}
  ]
}
```

Table 18: HTTP API – JSON api/state

The JSON data string is composed of 1 array named “states”. This array is filled with object representing a SCANeR module state. A SCANeR module state object is composed by the module name (as the object name) and module state string (as the object value). The SCANeR™studio GUI appear as “STUDIO”, it has an array of module state string (as the object value) and can have special state (see later).

The module state string can be:

String	State
DEAD	The module is in DEAD state
DAEMON	The module is in DAEMON state
LOADED	The module is in LOADED state
PAUSED	The module is in PAUSED state
READY	The module is in READY state
RUNNING	The module is in RUNNING state

Table 19: HTTP API – JSON api/state – module state

For SCANeR™ studio the “pending state” and “pending orders” are available:

String	State
LAUNCHING	Studio is starting the modules.
LAUNCHED*	The modules are all started.
LOADING	The modules are loading a scenario
LOADED	The modules are loaded.
STARTING	The module are going to the PAUSE state.
PAUSED	The modules are paused.
INIT	The module are going to the READY state.
READY	The module is in READY state
GO	The modules are going to the RUNNING state.
RUNNING	The module is in RUNNING state.
OPENING*	Studio is opening a scenario.
CONFIGURATION*	The configuration is changing.

Table 20 HTTP API – JSON api/sate – studio state

Studio can have multiple states.

- State with an * can exist with a state with no *.
- State with no * cannot be exist with other state with no *.
- Example: [“LAUNCHED” “OPENING”] if all modules are launched and Studio is opening a scenario.

IX.A.5.iii - api/configurations

```
{"configurations": [
    "Configuration name1",
    "Configuration name2",
    ...
    "Configuration nameN"
]} 
```

Table 21: HTTP API – JSON api/configurations

The JSON data string is composed of 1 array named “configurations”. This array is filled with all the SCANeR configurations found. Each configuration is represented as a string filled with the configuration name.

IX.A.5.iv - api/path?name=data

```
{"paths": [  
    "path name1"  
    "path name2"  
    ...  
    "path nameN"  
]
```

Table 22: HTTP API – JSON api/path?name=data

The JSON data string is composed of 1 array named “paths”. This array is filled with all the SCANeR paths founded in the current configuration. Each path is represented as a string filled with the path. The path itself can contain \${STUDIO_PATH}. This information signification is “relative to the STUDIO_PATH environment variable”.

IX.A.5.v - api/records

```
{"records": [  
    "Record name1"  
    "Record name2"  
    ...  
    "Record nameN"  
]
```

Table 23: HTTP API – JSON api/records

The JSON data string is composed of 1 array named “records”. This array is filled with all the SCANeR records found in the current configuration. Each record is represented as a string filled with the record name.

IX.A.5.vi - api/time

```
{"time": time in second}
```

Table 24: HTTP API – JSON api/time

The JSON data string is composed of 1 field. This filed is as floating number representing the time in second, with any number of decimal.

IX.A.5.vii - Correspondence with SCANeR™ studio GUI

Function	Studio GUI	HTTP API
Open a scenario	File/Open	/api/scenarios /api/open?scenario= <i>name</i>
Play a scenario	Play button	/api/play?scenario= <i>name</i>
Check the launch state	IHM	/api/state
Cancel launch	IHM Hit cancel Button	/api/cancel
Stop a simulation	Stop button	/api/stop
Pause a scenario	Pause button	/api/pause
Auto launch module	Automatically done	Automatically done
Launch default selected modules	Button start all	/api/start
Kill launched modules	Button stop all	/api/kill

Table 25: HTTP API – scenario management JSON vs SCANeR™ studio GUI

Function	Studio GUI	HTTP API
Select a config	Configuration/ Configuration Manager	/api/configurations
Change configuration	Configuration/ Configuration manager	/api/configuration?name= <i>config_name</i>
Edit/View data path	Configuration/Set datapath	View only : /api/path?name= <i>data</i>

Table 26: HTTP API – configuration management JSON vs SCANeR™ studio GUI

Function	Studio GUI	HTTP API
RePlay a record	Mode analyze, play gz	/api/records /api/play?record=name
Open AnalysingTool with a gz	Mode analyze, analyse a gz	/api/open?record=name
Change time of a record	Analyzing Tools, time line	/api/time (AnalysingTools API)
Pause/stop a record	Analyzing Tools, button pause/stop record.	/api/play (AnalysingTools API) /api/pause (AnalysingTools API) /api/stop (AnalysingTools API)
Analyze a record	Mode analyze, button analyze	/api/open?record=name (AnalysingTools API)
Close a record in AnalysingTool	Analysing Tool / right click on a record / unload	/api/close?record=name (AnalysingTools API)
Exit AnalysingTool	Analysing Tool/file/exit	/api/quit (AnalysingTools API)

Table 27: HTTP API – scenario management JSON vs AnalysingTool GUI

IX.B - SAMPLES

Here are two basic samples. They show all of the URI messages available. Feel free to program the behaviour you need by changing the source code.



The GUI windows requesting an answer can block the use of the remote controllers.

IX.B.1 - SIMPLE STUDIO CONTROLLER HTML5 SAMPLE

This sample is a little web application which acts as a SCANeR studio supervisor.

It used the described HTTP API.

It is just a few lines (about 130) of HTML and javascript. Delivered in the <STUDIO_PATH>\<STUDIO_VERS>\data\DEFAULT\supervision\index.html

It demonstrates all the Supervision HTTP api:

- Get the scenarios list in the active configuration
- Get the records list
- Open a scenario/record
- Play/pause stop a scenario/ a record
- Open a record in AnalysingTool
- Get the configurations list
- Change a configuration



Once the configuration is changed, press F5 to refresh the scenarios and records lists

- Get the module state
- Start/stop the simulators (the modules used by the simulator)

This sample is included in SCANeR studio default installation (Data/Default/Supervision). To launch it:

- launch SCANeR studio on a PC
- open on the SAME PC a browser at the address "<http://localhost:9090>"

Look the address bar, here the sample is connected to a SCANeR™ studio launched on the same machine (localhost).

The screenshot shows a web browser window with the following details:

- Address Bar:** http://localhost:9090
- Content Area:**
 - Configuration:** A dropdown menu set to "1.3_adas". A tooltip indicates it's filled via the API endpoint /api/configurations.
 - Change configuration:** A button labeled "Change configuration (api/configuration URI)".
 - Get paths used by Studio:** Buttons for "data" and "Path (api/path URI)".
 - Scenario:** A dropdown menu set to "toto2". A tooltip indicates it's filled via the API endpoint /api/scenarios.
 - Load into Studio:** A button labeled "Load into Studio (api/open URI)".
 - Record:** A dropdown menu set to "toto2-16_12_2015-12h11m09s". A tooltip indicates it's filled via the API endpoint /api/records.
 - Control Buttons:**
 - Play (api/play URI)
 - Pause (api/pause URI)
 - Stop (api/stop URI)
 - State Management:**
 - Get State (api/state URI)
 - Start Simulator (api/start URI)
 - Stop simulator (api/stop URI)

Illustration 68 : HTTP API – studio web monitor

You can also connect to SCANeR Studio from another computer.

Change "localhost" to the machine name or ip address where SCANeR studio GUI runs. The

port remains the same "9090".

The source code:

```
<html>
<head>
<meta http-equiv="content-type" content="text/html">
</head>
<script>
// function to populate the scenario drop list from the scenario list send by SCANeR studio
function populateScenario() {
    var xhr_object = null;

    xhr_object = new XMLHttpRequest();
    xhr_object.open("POST", "/api/scenarios", false);
    xhr_object.overrideMimeType('application/json');
    xhr_object.send(null);
    if(xhr_object.readyState == 4)
    {
        combo = document.getElementById("scenarios");
        scenarios = JSON.parse(xhr_object.responseText);
        for(var i = 0; i < scenarios.scenarios.length; i++) {
            var opt = scenarios.scenarios[i];
            el = document.createElement("option");
            el.textContent = opt;
            el.value = opt;
            combo.appendChild(el)
        }
    }
}

// function to populate the configuration drop list from the configuration list send by SCANeR studio
function populateConfiguration() {
    var xhr_object = null;

    xhr_object = new XMLHttpRequest();
    xhr_object.open("POST", "/api/configurations", false);
    xhr_object.overrideMimeType('application/json');
    xhr_object.send(null);
    if(xhr_object.readyState == 4)
    {
        combo = document.getElementById("configurations");
        scenarios = JSON.parse(xhr_object.responseText);
        for(var i = 0; i < scenarios.configurations.length; i++) {
            var opt = scenarios.configurations[i];
            el = document.createElement("option");
            el.textContent = opt;
            el.value = opt;
            combo.appendChild(el)
        }
    }
}

// function to populate the record drop list from the record list send by SCANeR studio
function populateRecords() {
    var xhr_object = null;

    xhr_object = new XMLHttpRequest();
    xhr_object.open("POST", "/api/records", false);
    xhr_object.overrideMimeType('application/json');
    xhr_object.send(null);
    if(xhr_object.readyState == 4)
```

```
{  
    combo = document.getElementById("records");  
    scenarios = JSON.parse(xhr_object.responseText);  
    for(var i = 0; i < scenarios.records.length; i++) {  
        var opt = scenarios.records[i];  
        el = document.createElement("option");  
        el.textContent = opt;  
        el.value = opt;  
        combo.appendChild(el)  
    }  
}  
</script>  
<body onload="populateScenario(); populateConfiguration(); populateRecords()">  
<h1>SCANeR Studio web monitor</h1>  
<hr>  
<form action="api/configuration" method="post"> configuration name :  
<select id="configurations" name="name"></select> This combo box is filled using the  
api/configurations URI  
<br>  
<input value="Change configuration (api/configuration URI)" type="submit">  
</form>  
  
<form action="api/path" method="post"> Get the paths used by Studio  
<input type="text" name="name">  
<input value="Path (api/path URI)" type="submit">  
</form>  
<hr>  
<form action="api/open" method="post"> scenario name :  
<select id="scenarios" name="scenario"></select> This combo box is filled using the  
api/scenarios URI  
<br>  
<input value="Load into Studio (api/open URI)" type="submit">  
</form>  
  
<form action="api/open" method="post"> record name :  
<select id="records" name="record"></select> This combo box is filled using the api/records URI  
<br>  
<input value="Open AnalysingTool (api/open URI)"  
type="submit" onclick="this.form.action='api/open'">  
<input value="RePlay record (api/play URI)" type="submit" onclick="this.form.action='api/play'">  
</form>  
  
<form action="api/play" method="get">  
<input value="Play (api/play URI)" type="submit">  
</form>  
  
<form action="api/pause" method="get">  
<input value="Pause (api/pause URI)" type="submit">  
</form>  
  
<form action="api/stop" method="get">  
<input value="Stop (api/stop URI)" type="submit">  
</form>  
<hr>  
<form action="api/state" method="get">  
<input value="Get State (api/state URI)" type="submit">  
</form>  
  
<form action="api/start" method="get">  
<input value="Start Simulator (api/start URI)" type="submit">  
</form>
```

```
<form action="api/kill" method="get">
<input value="Stop simulator (api/kill URI)" type="submit">
</form>

</body>
</html>
```

IX.B.2 - SIMPLE ANALYSIS CONTROLLER HTML5 SAMPLE

This sample is a little web application which controls the AnalysingTool application.

It uses the described Analyse HTTP API.

It is just a few lines (about 64) of HTML and javascript. Delivered in the
`<STUDIO_PATH>\<STUDIO_VERS>\data\DEFAULT\analyze\index.html`

It demonstrates all the Analyse HTTP api:

- Open a record
- Close a record
- Play/pause stop a record
- Change the record time
- Get the record time
- Quit the application.

This sample is included in SCANeR™ studio default installation (Data/Default/Analyze). To launch it:

- launch AnalysingTool on a PC
- open a browser on the SAME PC and enter the address "<http://localhost:9091>"

Look the address bar, here the sample is connected to an AnalysingTool launch on the same machine (localhost) but it can also connect to an AnalysingTool launch on another machine. Just change the "localhost" to the machine name or ip address. The port remains the same "9091".

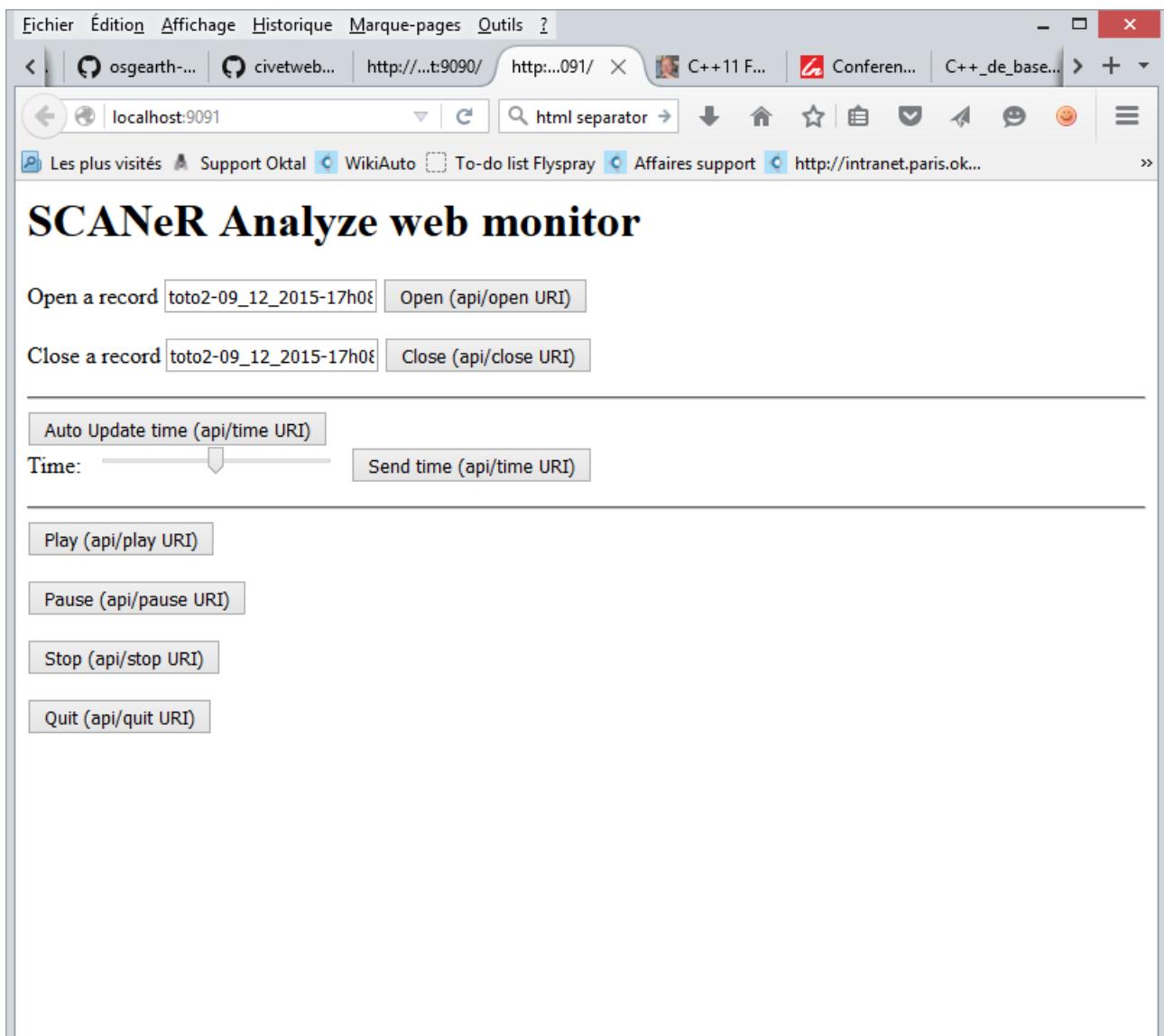


Illustration 69 : HTTP API – Analyse web monitor

The source code:

```

<html>
<head>
<meta http-equiv="content-type" content="text/html">
</head>
<script>
function updateTime() {
    var xhr_object = null;

    xhr_object = new XMLHttpRequest();
    xhr_object.open("POST", "/api/time", false);
    xhr_object.overrideMimeType('application/json');
    xhr_object.send(null);
    if(xhr_object.readyState == 4)
    {

```

```
        range = document.getElementById("time");
        time_res = JSON.parse(xhr_object.responseText);
        range.value = time_res.time
    }
}

function forevertime()
{
    setInterval(function(){updateTime()}, 100);
}

</script>
<body>
<h1>SCANeR Analyze web monitor</h1>
<form action="api/open" method="get"> Open a record
<input type="text" name="record" value="toto2-09_12_2015-17h08m18s">
<input value="Open (api/open URI)" type="submit">
</form>

<form action="api/close" method="get"> Close a record
<input type="text" name="record" value="toto2-09_12_2015-17h08m18s">
<input value="Close (api/close URI)" type="submit">
</form>
<hr>
<button onclick="forevertime()">Auto Update time (api/time URI)</button>

<form action="api/time" method="get">
    Time:
    <input id="time" type="range" name="time" min="0" max="10" step="0.1">
    <input type="submit" value="Send time (api/time URI)">
</form>
<hr>
<form action="api/play" method="get">
<input value="Play (api/play URI)" type="submit">
</form>

<form action="api/pause" method="get">
<input value="Pause (api/pause URI)" type="submit">
</form>

<form action="api/stop" method="get">
<input value="Stop (api/stop URI)" type="submit">
</form>

<form action="api/quit" method="get">
<input value="Quit (api/quit URI)" type="submit">
</form>

</body>
</html>
```

x - VEHICLE DYNAMICS CALLAS API

x.a - PRESENTATION

Vehicle Dynamics Callas API allows to create an external model. An external model is a third party DLL¹⁰ that can be loaded by the SCANeR™ studio auto-simulation software to:

- Add features (real-time driving commands, etc.)
- Replace one part of the vehicle model (engine, suspension etc.) with a different software implementation or even a hardware implementation (hardware in the loop) such as an engine bench or an ABS electronic box.

In the “External modules” tab within “Utilisation parameters”, the user can define settings for all external models, including:

- C++ DLLs that model a feature that is not a vehicle component
- C++ DLLs that model a vehicle component during the development stage
- SFonctions during the co-simulation with Matlab/Simulink (offline simulation only)
- DLLs used to communicate using other protocols¹¹



External modules directory is not used here. Selected DLLs can be located in any directory. However, we highly recommend you to store all your externals models in <STUDIO_PATH>\<STUDIO_VERS>\data\<MYCONF>\vehicle\Callas\Externals

x.b - DEFINITION

x.b.1 - MODEL TYPES

x.b.1.i - Organic model

These modules substitute part of Callas Vehicle model for another model developed by the user. This applies to engine, steering, suspension models, etc.

x.b.1.ii - Functional model

These modules add an extra layer to the existing model, without replacing any of its parts. With the GMP Torque module for instance, the user can add another coupling to the engine, clutch and/or gearbox output while the equivalent Vehicle Dynamics Callas models are enabled.

¹⁰Dynamic Dink Library

¹¹ HLA, etc.

X.B.2 - MODEL FUNCTIONS

A model can export a number of “functions”. A function is defined by a group of output values. Get the functions description into the **VehicleDynamicsCallasInputs.html** document (menu **HELP / Getting started** / **VehicleDynamicsCallasInputs.html** or or **<STUDIO_PATH>\<STUDIO_VERS>\doc\VehicleDynamicsCallasInputs.html**).



In the case of the “gearbox command” function, the group of values is in fact a single value: the gear ratio number. Please refer to the appendix for the list of all available functions.

X.B.3 - MODEL CHANNELS

“Channels” refer to the values computed by the SCANeR™ studio vehicle model. They can be accessed through the external model. The list of channels is in constant evolution; it can be displayed by clicking on the “**HELP / Getting started / VehicleDynamicsCallasOutputs.html**” menu (or **<STUDIO_PATH>\<STUDIO_VERS>\doc\VehicleDynamicsCallasOutputs.html**).

X.B.4 - MODEL MEMBERS

The members associated to an external module can either be of the “int” or “double” type, or any other type as defined by the OKTAL team:

Structure	Attributes	Description
SPoint2D	double X, Y;	X and Y coordinates of a point in 2 Dimensions
SPoint3D	double X, Y, Z;	X, Y and Z coordinates of a point in 3 Dimensions
SMark	double X, Y, Z; double roll, pitch, heading;	Enables definition of a spatial reference by using the coordinates of its origin relative to the global spatial reference, and the definition of the rotation of its 3 axes (in roll, pitch and yaw)
STorser	SPoint3D PointApplication; SPoint3D Force; SPoint3D Moment;	A torsor being applied to a solid is defined by: <ul style="list-style-type: none">● its point of application in the spatial reference linked to the solid expressed in the dynamic coordinate system, according to the SAE convention (http://www.sae.org) : centered on the center of gravity● a force field (independent from the point of application)● a moment field which can be expressed differently, depending on the point of application.
RoadQueryVal	SPoint3D pos SPoint3D norm int laneType	

	int groundIndex	
InitCondition	SPoint2D Pos double Heading double Speed double FuelTankRatio	
InfoCalcul	int nbvhc C_PATH NameTerrain char rab[148]	
VhcData		

Members can also be tables of elementary types.

X.B.4.i - Table Index Conventions

Index conventions must be respected for member tables, to avoid using erroneous values.

Index conventions used are as follows :

- by axle: one dimension assigned to the number of axles (#0 index = #1 axle)
- by wheel: two dimensions
 - the first dimension is assigned to the number of axles: #0 index = #1 axle
 - the second is assigned to the side the wheel is on: #0 index = left side, 1 index = right side
- by tyre: three dimensions
 - the first dimension is assigned to the number of axles: #0 index = #1 axle
 - the second is assigned to the side the tyre is on:
 - #0 index = left
 - #1 index = right
 - the third dimension is assigned to the tyre:
 - #0 index = normal tyre (always on the inside)
 - #1 index = twinned tyre (always on the outside)
- by roller: two dimensions
 - the first dimension is assigned to the number of rollers
 - the first index must relate to the pulley or sprocket
 - the last index used must complement the first index (pulley or sprocket). The choice of index varies depending on the element geometric position: if the pulley is positioned at the front of the vehicle, the data will be saved in the 0 index. It will otherwise be saved in the last used index.
 - the second dimension is assigned to the side the roller is on:
 - #0 index = left

- #1 index = right

Every torsors and forces are expressed in the dynamic coordinate system, according to the SAE convention (<http://www.sae.org>): centered on the center of gravity,.

The architect frame is not used in this case. As the position of the COG is not an input, for knowing it, you have to use EG file.

x.c - HOW IT WORKS

x.c.1 - INITIALISATION

When the SCANeR™ studio computation initializes, it loads the external models that have been set up and recovers their characteristic data, including:

Their internal name to be printed in the log to confirm it has been correctly loaded.

The list of channels it needs as inputs

The functions it is required to export (its output)

The time at which the model should be called.

x.c.2 - COMPUTATION LOOP

At each step during a computation loop, the input channels are sent to the external module. The integration function is called, mentioning the time elapsed since the last call. The model's output is then updated to be passed onto SCANeR™ studio.

x.d - VEHICLE DYNAMICS CALLAS API WITH C++

x.d.1 - INTRODUCTION

This chapter is aimed at developers who wish to develop their own external modules for the Vehicle Dynamics Callas model. It assumes the user has some working knowledges of Vehicle Dynamics Callas modeling, as well as some C or C++ programming experiences. Some of the information given here is specific to the Microsoft Visual C++ programming environment. Equivalents exist for other environments; however, they are not discussed here.

The "APIs" sub folder contains the files necessary to get started:

- "Include": folder containing the ".h" header files that define the interface between the modules and Callas Dynamic model
- Examples folder containing sample external modules:
`<STUDIO_PATH>\<STUDIO_VERS>\APIs\samples\VehicleDynamicsCallas`
 - [ExternalDefects](#), description: X.I.1 - ExternalDefects sample - 188
 - [SimpleEngine](#), description: X.I.3 - SimpleEngine sample - 188

- SinusPilot, description: Erreur : source de la référence non trouvée - Erreur : source de la référence non trouvée - Erreur : source de la référence non trouvée
- SinusPilot_c, description: X.I.8 - SinusPilot_C - 191

These examples are commented and all functions explained. A number of functions are not detailed in the text documentation.



The latest service pack for visual studio should be installed.

X.D.2 - SETTING UP THE DEVELOPMENT ENVIRONMENT

The SDK was tested using compiler described in chapter IV.B - Environment - 18, but it should work with other C/C++ environments. The settings described below are applied to Visual Studio but should be similar when using make/gcc or Borland C++, etc.

Setting up the "include" directory

The SCANeR™ studio “include” folder is specified once and for all, and for all external models, in the Visual Studio options.

In Visual C++, simply add the location of the SCANeR™ studio “include” directory to the list. This way, the “include” files will always be up to date when upgrading SCANeR™ studio.

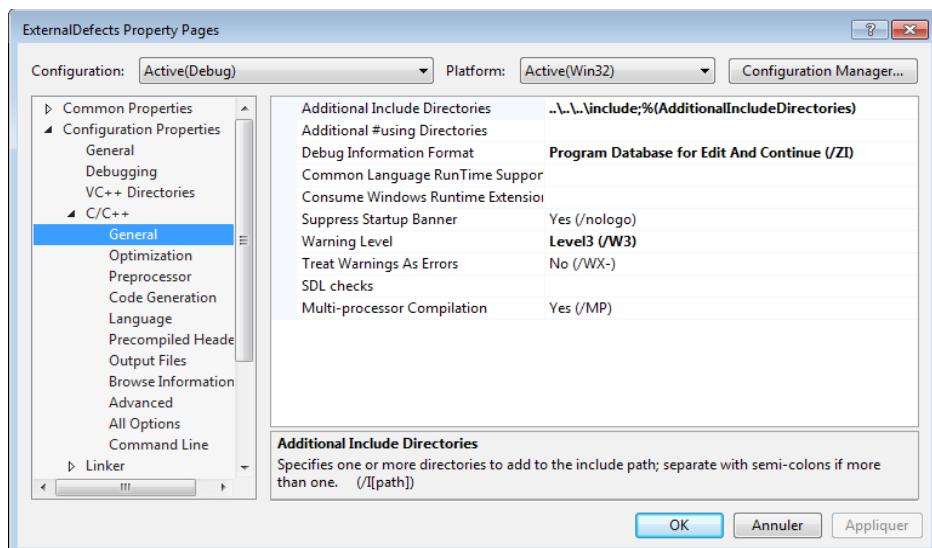


Illustration 70 : Include environment



By default, the location for these “Include” files is in "`<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\VehicleDynamicsCallas`".



The header files contained in “APIs/include” should not be copied into the external model’s project.

X.D.3 - SAMPLES COMPILATION

Get the needed third party software according to your needs, for more details refer to chapter IV.B - Environment - 18.

Open the “complete.sln” solution located under <STUDIO_PATH>\<STUDIO_VERS>\APIs\samples and build VehicleDynamicsCallas samples.

To learn more about VehicleDynamicsCallas samples provided with the SCANeR™ studio environment, refer to X.I - ANNEXE 1: VehicleDynamicsCallas Samples in SCANeR™ studio environment - 188

X.D.4 - MAKE A NEW EXTERNAL PROJECT

X.D.4.i - Copy an example

To create a new external model, the simplest way is to copy one of the samples into a user directory. Choose an appropriate sample to copy: it should closely match your model-building objective. Also, you can choose between a C and a C++ sample.

X.D.4.ii - Rename the project

To avoid confusion, rename the project. Renaming a “Visual Studio” project is not trivial, so choose the new name carefully to avoid repeating the operation later. The name of the project should indicate what the model will be used for.



A model that applies braking torque at the wheels in order to correct a trajectory, and developed by the “RailTech” company could be called: “ESPRailTech”.

To rename a project, first rename the folder that contains it, then the “.h .c .cpp .sln” and “.vcproj” files.

X.D.4.iii - Implement your new functions

By using the sample code as an example and reading the detailed comments, replace the code in the functions with what you need to implement in your own model.

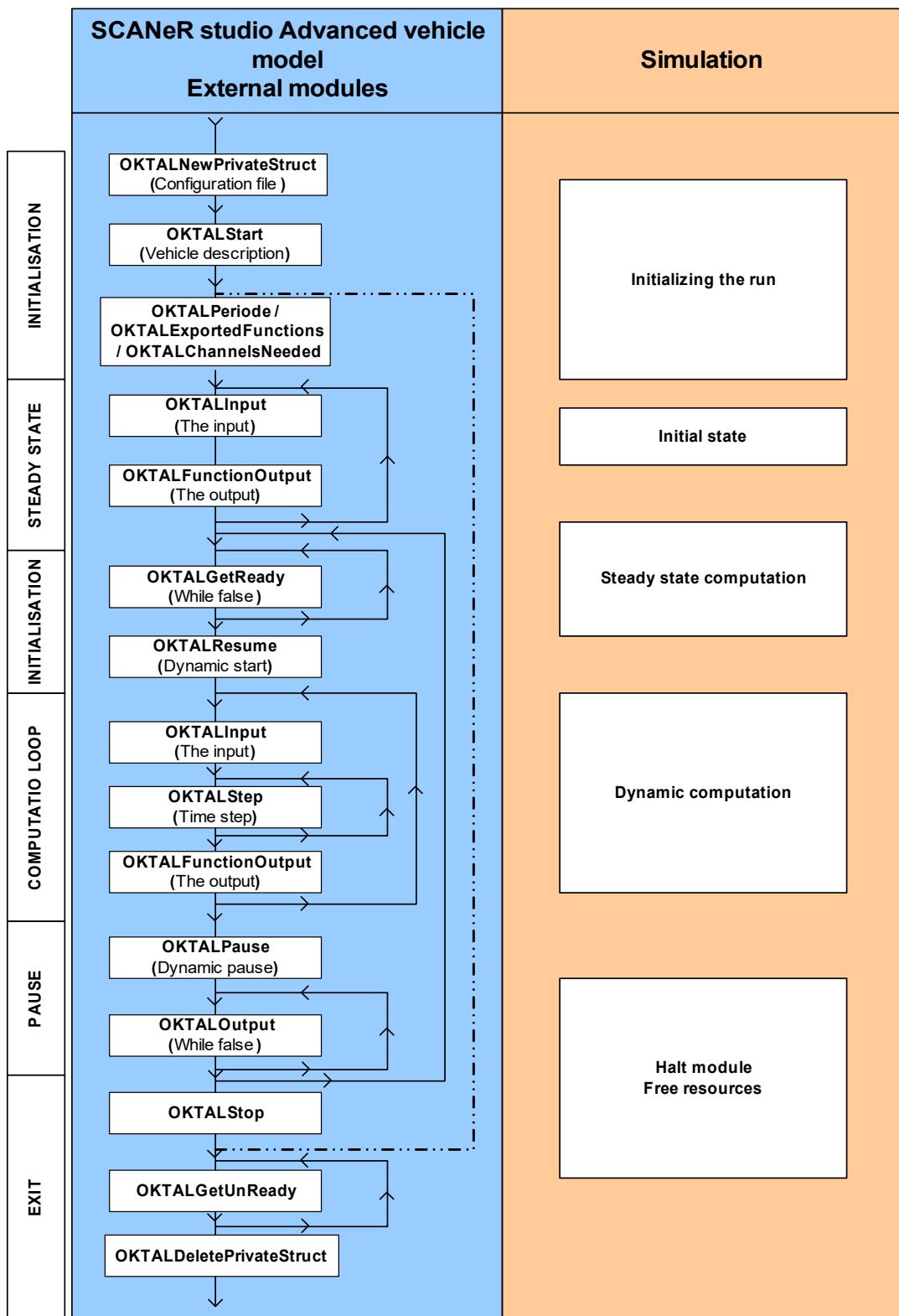
X.D.4.iv - Test often

It is preferable to test that the compiledVehicleDynamicsCallas dll still works as an external model at all stages of the project. The project should also be backed up on a regular basis, by copying the project folder and renaming it with a different version number.



It is a good idea to enter different version numbers for each back-up version, such as: “ESPRailTech_1 ESPRailTech_2 ESPRailTech_3”, while keeping the same filename for the current development version “ESPRailTech”. This avoids having to change the SCANeR™ studio path or the debugger each time a new version number is saved.

X.D.5 - FUNCTION CALL ORDER



X.D.6 - USING THE EXTERNAL MODEL

X.D.6.i - Setting up the external model

An external model needs to be set up in SCANeR™ studio before it can be tested. To setting up an external model refer to chapter X.G - Use an external model on a Vehicle Dynamics Callas - 183.

X.D.6.ii - The configuration file

During initialization, the path to the external model configuration file is passed onto the OKTALNewPrivateStruct() function.

The external model developer selects the configuration file format. SCANeR™ studio does not read or write this file which can contain information necessary to initialise the external model.

The “ESPRailTech” configuration file could contain various thresholds and options that need to be tuned for each vehicle for instance.

Optionally, the model can contain a OKTALEditConfig() function. This function should present the user with a dialog box enabling him to edit the configuration path that was passed on to the function.

X.D.6.iii - Launching a configuration

Once the external model has been set up either in the vehicle or in the utilisation parameter, any computation can be used to test it. Once the run is over, read the computation’s log file and check for lines referring to the external module. If the module was correctly loaded, a description of the model will be written in the log.

X.D.6.iv - Debugging

To debug an external model:

- Set the path to the DLL debug version (and not to the release version).
- Start the simulation from SCANeR™ studio user interface so that temporary files are created. As soon as the run starts, it can be aborted.

In your project settings, choose the “CalculModulaire.exe” file to launch the debug session.

X.D.7 - MAINTAINING AN EXTERNAL MODEL

X.D.7.i - Changing SCANeR™ studio release numbers

When upgrading your SCANeR™ studio version, neither the path to the SCANeR™ studio SDK nor the header files will change. If you did not add the header files to your project, all you have to do is recompile and your model will be compatible.

X.D.7.ii - Changing SCANeR™ studio versions

When upgrading your SCANeR™ studio version, you will need to point to the correct “Include” location in your Visual Studio settings (as described in chapter 4), before recompiling. Please note that additional modifications might be needed for your code to work with big upgrades.

X.D.7.iii - Compatibility

SCANeR™ studio is developed attempting to avoid binary or source code incompatibilities between external model dll's. The only way to ensure a given model will work is by checking if it was compiled with the header files of the SCANeR™ studio version it is to be used with.

X.D.8 - TIPS AND TRICKS

When the code in an external model raises an exception (segmentation fault, floating point overflow etc.) it will cause the SCANeR™ studio computation or interface to quit unexpectedly. Mixing Release and Debug executables with DLLs can also cause unexpected exits. In all cases, check that the external model loads correctly by verifying if the model “internal name” is displayed in both the SCANeR™ studio interface and in the computation log. The internal name is queried by the OKTALName() function.

X.E - VEHICLE DYNAMICS CALLAS API WITH MATLAB\SIMULINK

X.E.1 - INTRODUCTION

This chapter is aimed at developers who wish to develop their own external modules for the Vehicle Dynamics Callas model. It assumes the user has some working knowledges of Vehicle Dynamics Callas modeling, as well as some MatLab\Simulink programming experiences. S

The “APIs” sub folder contains the files necessary to get started:

- **simulink\VehicleDynamicsCallas**,
- **samples\VehicleDynamicsCallas\Simulink**: Examples folder containing sample external modules:
 - ExternalDefect, description X.I.2 - Matlab_ExternalDefectst sample - 188
 - SimpleEngine, description X.I.4 - MatlabSimpleEngine sample - 188
 - SimpleEngine_rtwWrapper, description X.I.5 - MatlabSimpleEngine_rtwWrapper sample - 189
 - SinusPilot, description X.I.7 - MatlabSinusPilot sample - 191

X.E.2 - FIRST STEPS

X.E.2.i - MatLab path

Before running the first co-simulation on a PC, a number of settings must be made to enable communication between MatLabSimulink and Vehicle Dynamics Callas model.

First, MATLAB's path must be set so the programme can find:

- the mex file for the S-Function that communicates with Vehicle Dynamics Callas model:
`<STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>`
- the Simulink library specific to communication with Vehicle Dynamics Callas model:
`<STUDIO_PATH>\<STUDIO_VERS>\APIs\simulink\VehicleDynamicsCallas`

When clicking on MatLab's **Files/Set path** menu, the following dialog box is displayed:

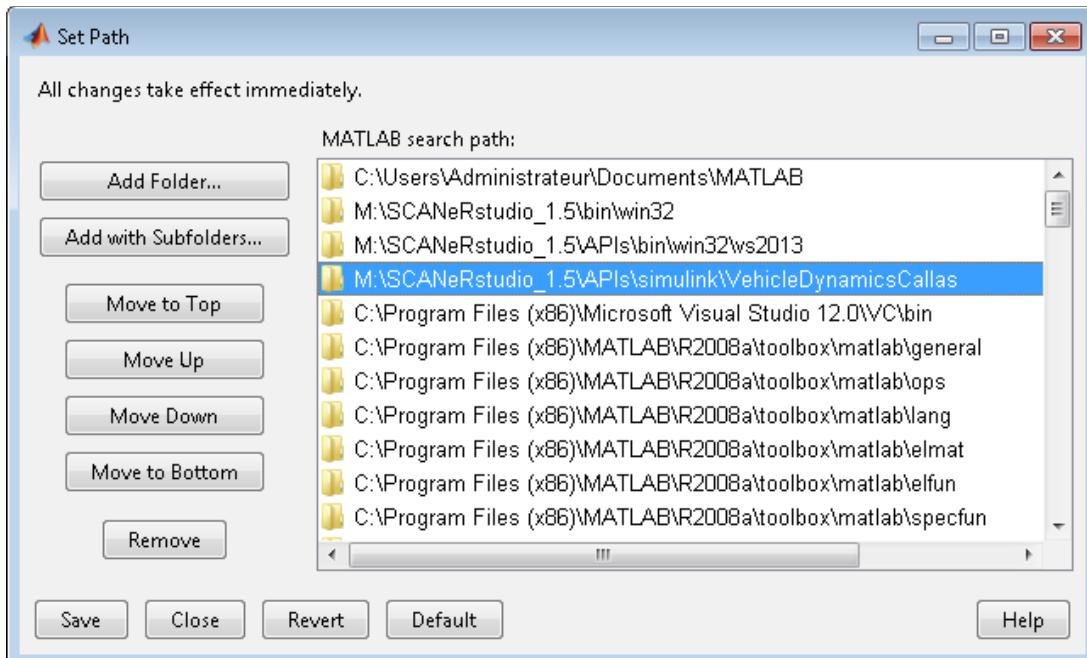


Illustration 71 : MatLab set path



If the user decides to store the OKTAL_CallasPlugin.mdl and slblocks.m files (these two files cannot be separated from one another) in a different location, the destination folder will then have to be added to the MatLab Path.

X.E.2.ii - Compiler configuration

To build executable code from the Simulink model, it is necessary to have the compiler described in chapter IV.B - Environment - 18 installed on the computer. The Matlab command "mex -setup" allows selecting the compiler.

X.E.3 - SIMULINK BLOCKS

X.E.3.i - Inputs

Refer to X.B.2 - Model functions - 147.

The choice of the externalized functions is made only by connecting an Input_Blocks function with the corresponding input in vehicle model

The entries whose functions won't be externalized must **IMPERATIVELY** stay unconnected.



SCANeR™studio only externalizes the functions whose the model vehicle entries are connected. All the others necessary variables will be directly calculated by the Callas model model.

X.E.3.ii - Output

X.E.3.ii.a - Vehicle

```

1001_FnctdOutputData
1002_FnctdOutputFn
1003_FnctdOutputCommandSteering
1004_FnctdOutputCommandPedal
1005_FnctdOutputCommandTire
1006_FnctdOutputTire
1007_FnctdOutputBrake
1008_FnctdOutputBrakeEddy
1009_FnctdOutputForce
1010_FnctdOutputSolid
1011_FnctdOutputCp
1012_FnctdOutputCpSteering
1013_FnctdOutputWheelTorque
1014_FnctdOutputWind
1015_tireed
1016_FnctdOutputBrakePressure
1017_FnctdOutputGearboxCommand
1018_FnctdOutputSKid
1019_FnctdOutputCompositetrail
1020_FnctdOutputVehicleDynamics
1021_FnctdOutputCpCp
1022_FnctdOutputEddy
1023_FnctdOutputCommandPedals
1024_FnctdOutputVehicleSteering
1025_FnctdOutputBrakeEddy
1026_FnctdOutputBrakeComposeATrail
1027_FnctdOutputBrakeDynamics
1028_FnctdOutputCpCp
1029_FnctdOutputBrake
1030_FnctdOutputVehicleEddy
1031_FnctdOutputVehicleSuspensionWheel
1032_FnctdOutputVehicleSuspension
1033_FnctdOutputVehicleMatters
1034_FnctdOutputVehicleMatters
1035_FnctdOutputCpCp
1036_FnctdOutputGearbox
1037_FnctdOutputVehicleBodyPosition
1038_FnctdOutputVehiclePosition
1039_FnctdOutputVehicleForce
1040_FnctdOutputCpCp
1041_FnctdOutputCpCpTorque
1042_FnctdOutputVehicleRoll
1043_FnctdOutputVehicleTire
1044_FnctdOutputVehicleSuspension
1045_FnctdOutputVehicleSensor
1046_FnctdOutputGearboxCommandSip
1047_tireed
1048_FnctdOutputVehicleKey
1049_FnctdOutputToggleElectroBrakeInt
1050_FnctdOutputSuspensionInt
1051_FnctdOutputVehicleAppearance
1052_FnctdOutputVehicleCpCp
1053_FnctdOutputVehicleCpCp
1054_FnctdOutputVehicleHeight
1055_FnctdOutputVehicleCommand
1056_FnctdOutputVehicleSteeringTorq
1057_FnctdOutputVehicleSolid

```

Vehicle Model Output

Callas vehicle model

Illustration 72 : Vehicle block

This block contains the S-Function. It enables the communication with SCANeR™studio

Left inputs enable the vector variables computed by MatLab\Simulink to be plugged into the correct SCANeR™studio inputs, depending on the co-simulated function.

One of the inputs ("User data") is dedicated to receiving channels defined by the user, as fed by Simulink and stored in SCANeR™ studio's results file to allow easier post-treatment. It can include a maximum of 1023 channels.

It is important to notify that the order of the chosen channels and sent in "Custom data" must be the same that the configuration file giving the personalized Simulink model outputs

The output wire (right side) must be connected to a Demux so the values generated by SCANeR™ studio's computations can become Simulink inputs.

The output vector is defined by the user : it respects the channel order define in the configuration file giving the Eg channel in Simulink.

The choice of the externalized functions is made only by connecting an Input_Blocks function with the corresponding input in vehicle model

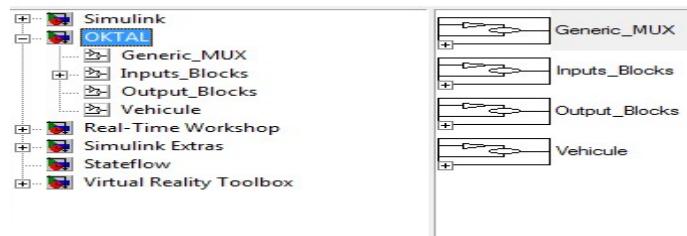


The entries whose functions won't be externalized must **IMPERATIVELY** stay unconnected. SCANeR™ studio only externalizes the functions whose the model vehicle entries are connected. All the others necessary variables will be directly calculated by the Callas model model.

X.E.3.iii - Simulink library

The Simulink library OKTAL is dedicated to SCANeR™ studio. it corresponds in Simulink at the SDK in C++ from SCANeR™ studio, the functions names and the variable names are the same. It is constituted by 4 parts :

- **Generic_MUX** : generic MUX used by the Co-simulation (This library must only be used in advanced use where the Input_Blocks library components are not sufficient)
- **Inputs_Blocks** : SDK functions. The 52 functions presented have in their library all the elements to simply co-simulate
- **Output_Blocks** : 2 Demux to connect at the vehicle model output. Each allows to use SCANeR™ studio variable in our Simulink model
- **Vehicule** : Sfunction representing the SCANeR™ studio vehicle model



X.E.3.iii.a - Inputs_Blocks

This library contains all the necessary elements to execute the 52 SDK functions.

For example : **013_FunctionOutputWheelTorser**

This function add the following elements :

	<p>Block to connect at vehicle on the input 013_FunctionOutputWheelTorser</p> <p>This block is preplanned to simplify when the model is a 4 wheels model.</p>
	<p>This block is the same as previously except that it is preplanned for a 12 wheels model. Moreover, we must know that in SCANeR™ studio, it is possible to have 10 axles. In this case, we must open this block and add the connections for all the additional axles.</p>
	<p>Use it to inform the torser X Y Z components (For example)</p>
	<p>Use it to apply a torser</p>

X.E.3.iii.b - Functioning synthesis

Colour code in the library

- **Darkness blue** : SCANeR™ studio Sfunction in correspondence to vehicle model
- **Orange** : in correspondence to the functions blocks to connect at vehicle model
- **Light blue** : in correspondence to the MUX allowing model to have the wished dimension in SCANeR™ studio. Generally, the MUX are preplanned for 4wheels vehicle and 12 wheels vehicle. This wheels can be twin wheels or not. For example, the function 021_FunctionOutputTireGrip is in charge of twin wheels.



Remark : when the preplanned MUX are not sufficient, you must open this one and complete the missing entries function of the numbers of axles that you want to add in model

- **White** : important block to complete the model

X.E.4 - Co-SIMULATION

X.E.4.i - Presentation

It is possible to use the Simulink model in a SCANeR simulation without compiling it.

Co-simulation involves modelling a Vehicle Dynamics Callas sub-function. SCANeR™ studio and MatLab\Simulink software programmes synchronously integrate their own state variables and exchange a number of variables.

Co-simulation can be used to model functions that are external to the vehicle (such as the pilot, ground geometry, etc.) or to model vehicle parts (engine, suspensions, etc.).

The fundamental differences between using co-simulation and using an external dll include:

- The two software programmes work in parallel: the two graphic user interfaces are open and can be edited at the same time. Synchronisation occurs when running a computation.
- The numerical method time step can be either imposed by MATLAB/Simulink on SCANeR™ studio (shared memory communication) or it can be free (Network communication).

In MatLab/Simulink, a specific block is provided representing the Vehicle Dynamics Callas. This block input is the data relating to the externalised function (channels that the Simulink model must provide to the dynamic model). Its output contains data relating to the vehicle state, generated by computations.

Channels output by the Vehicle Dynamics Callas block are defined in a file that follows Vehicle Dynamics Callas custom graph rules ("*.gra" files). Up to 1024 channels may be selected in Vehicle Dynamics Callas results file channel list.

X.E.4.ii - Data exchange methods

Vehicle Dynamics Callas is supplied with two mex files. These files enable two different co-simulation methods with MATLAB/Simulink. The co-simulation methods are:

- **Mem** functions: Can be run on the same computer as Vehicle Dynamics Callas model. Data are exchanged through shared memory. This is the fastest method available; data are exchanged at each step.
- **Net** functions: Enable “distributed” calculations: Vehicle Dynamics Callas model and MATLAB/Simulink run on different computers (not always). The data is exchanged through the TCP/IP protocol, but at a lower transfer rate.

These mex files are located under <STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>:

- SfonctionCallasMem.mexw32 (or SfonctionCallasMem.mexw64 for 64 bits)
- SfonctionCallasNet.mexw32 (or SfonctionCallasNet.mexw64 for 64 bits)



Mex files are available in 32 or 64 bits. Choose the same platform as your MatLab\Simulink.

	Advantages	Disadvantages	Application
Shared memory	<ul style="list-style-type: none"> - Easy to use (Vehicle Dynamics Callas model is controlled by Simulink) - data exchange at each time step - Simulink controls time steps 	<ul style="list-style-type: none"> - Using a single PC to run both programmes - Hardware must be powerful - When Simulink programme is complex (slow) but does not require small time steps, computation will be time-consuming, as Vehicle Dynamics Callas model requires small time steps, 	mechanical models
TCP/IP	<ul style="list-style-type: none"> - One single software programme per computer (computers can be remote) 	<ul style="list-style-type: none"> - Data is exchanged at constant real-time intervals, not at each time step. - Slower than shared memory - Co-simulation is more difficult to handle, with 2 pieces of software to simultaneously manage (and set up) on two different PCs. - Slows down local network 	control loops

In most cases, shared memory links are preferable, because synchronisation occurs at each time step. This is essential when a vehicle component is modelled. It can also be easier to work with 2 different applications running on the same computer.

The network link must be reserved for models requiring a great deal of resources and which do not need to be synchronised at each time step (i.e: control loops).

To allow the vehicle dynamics callas communication (SCANeR™ studio side) with MatLab\Simulink, the vehicle dynamics callas needs to be associate to the CallasSimulinkMemInterface.dll in its vehicle instance. It is located under <STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>\plugins\Callas.

X.E.4.iii - Limitations

All SCANeR™ studio computations can be run during co-simulations. However, MATLAB/Simulink does not know how to handle steady-state computations or computations requiring backward steps. It can only handle dynamic computations with strictly increasing time values.

SCANeR™ studio is then used during the steady-state initialisation that preceding the launch of a dynamic computation, and during a pure steady-state dynamic analysis. SIMULINK will only play a role during a dynamic computation.

To ease the transition between the initial state and the dynamic computation, SCANeR™ studio internal model must be set to react in a similar manner as the Simulink model.

X.E.4.iv - Run a Co-simulation

X.E.4.iv.a - Vehicle dynamics callas settings

1. Create a scenario and use a Callas vehicle.
2. Edit the vehicle instance, go to the tab **Callas** and edit the **External models** tab.

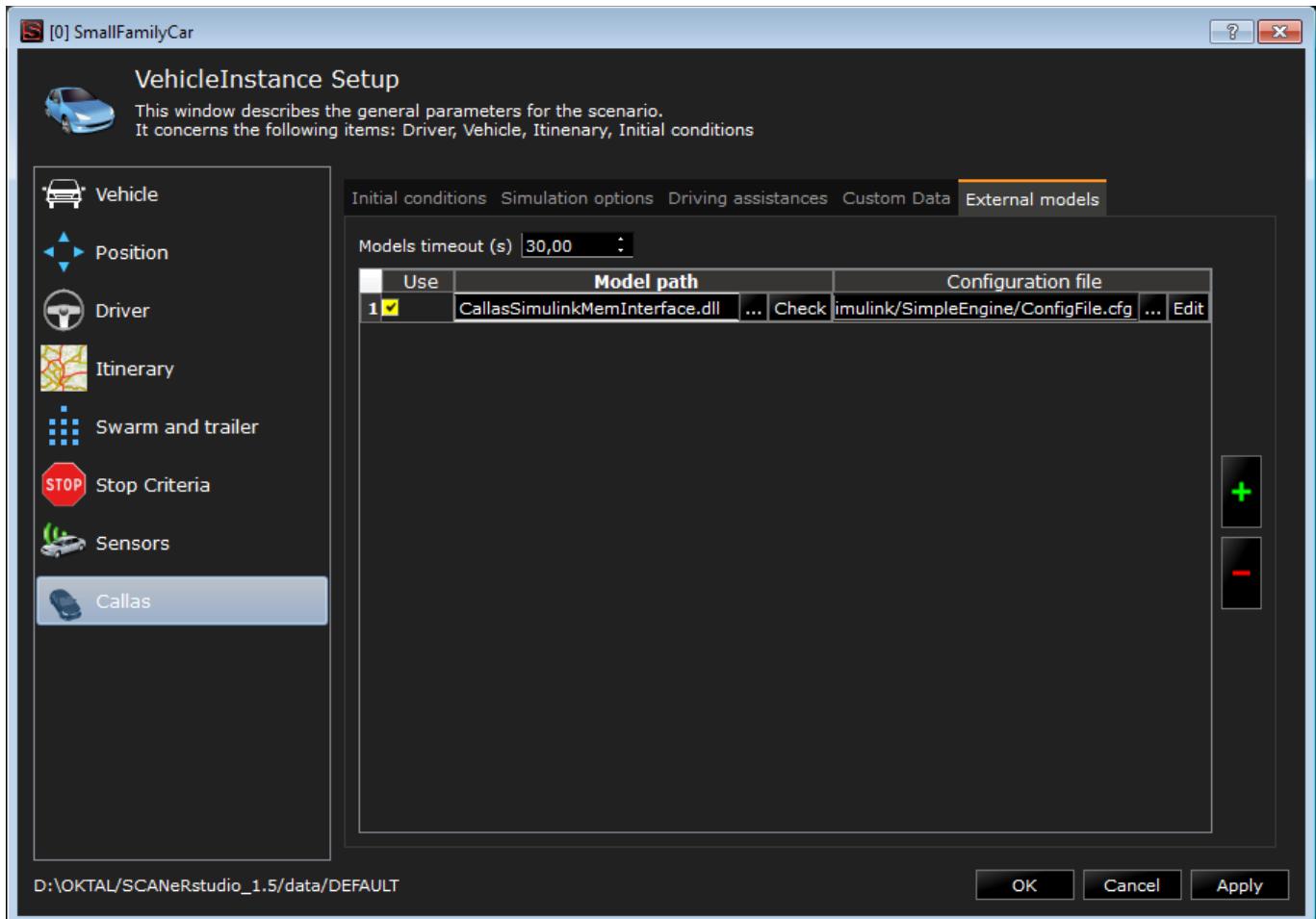


Illustration 73 : Callas instance, external models tab

3. Add a new external model by clicking on the “+” button and select in the:
 - Models timeout(s): Time-out at connection (maximum time given to the 2 software programmes to establish communication). A basic value is 30 seconds.
 - Model path: CallasSimulinkMemInterface.dll
 - Configuration file: Configuration file associated with module. It enables the user to set Callas model channels used by Simulink module. It provides the following informations:
 - EG Callas model channels used to Simulink model
 - Personalized Simulink model outputs available for consultation in SCANETM studio EG
4. To select the model to use, click on the button '...', the following window will appear
The model type depends of the data exchange method you plan to use, for:
 - **Mem (shared memory)** exchange select the “Dynamic Link Library (DLL)” option,

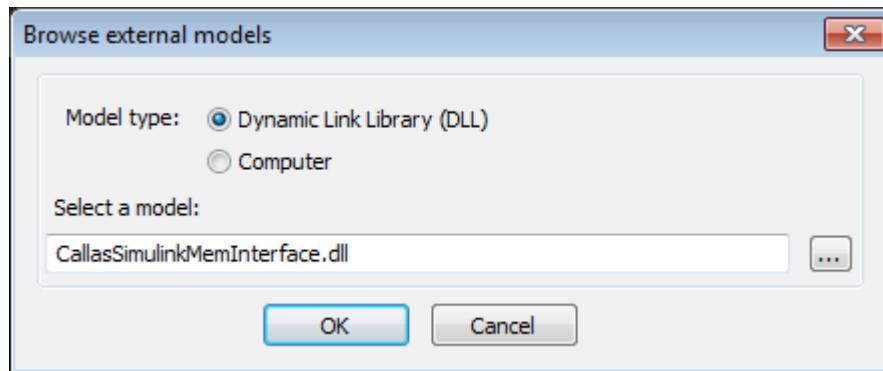


Illustration 74 : Shared memory communication



If the model that you choose is not under the folder <STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>\plugins\Callas, the absolute path needs to be filled, otherwise only the model name needs to be filled. The advantage of using the second solution is that SCANeR™ studio will choose for you the dll architure that corresponds to your MatLab\Simulink architeture (32 or 64 bits).

- o **Net (network communication)** exchange select the “Computer” option.

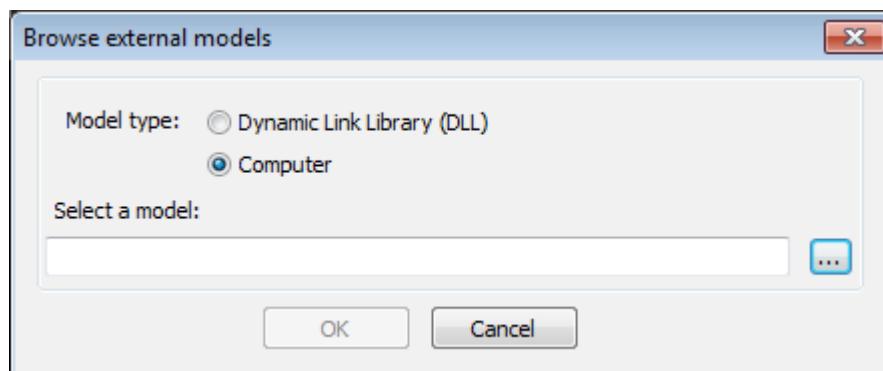


Illustration 75 : Network communication

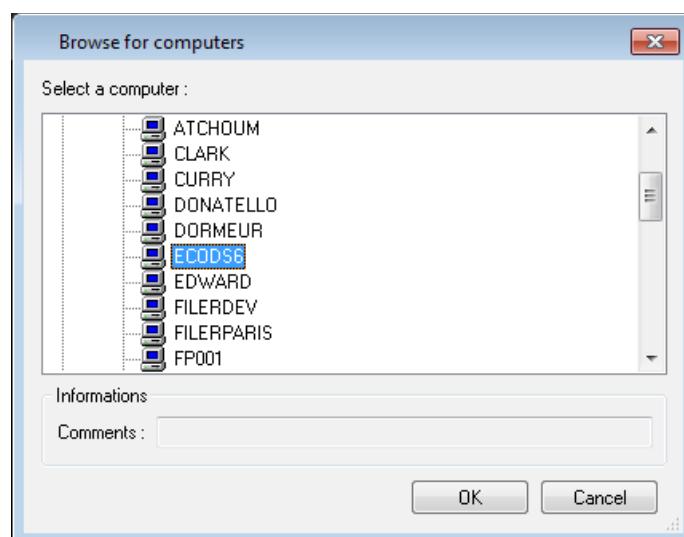


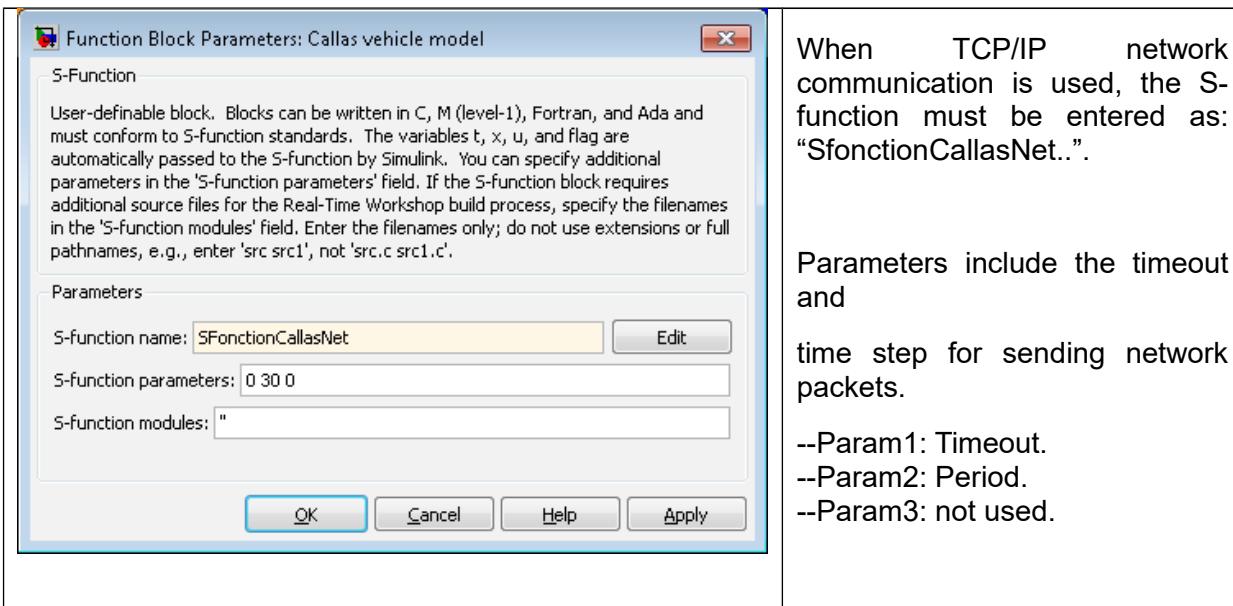
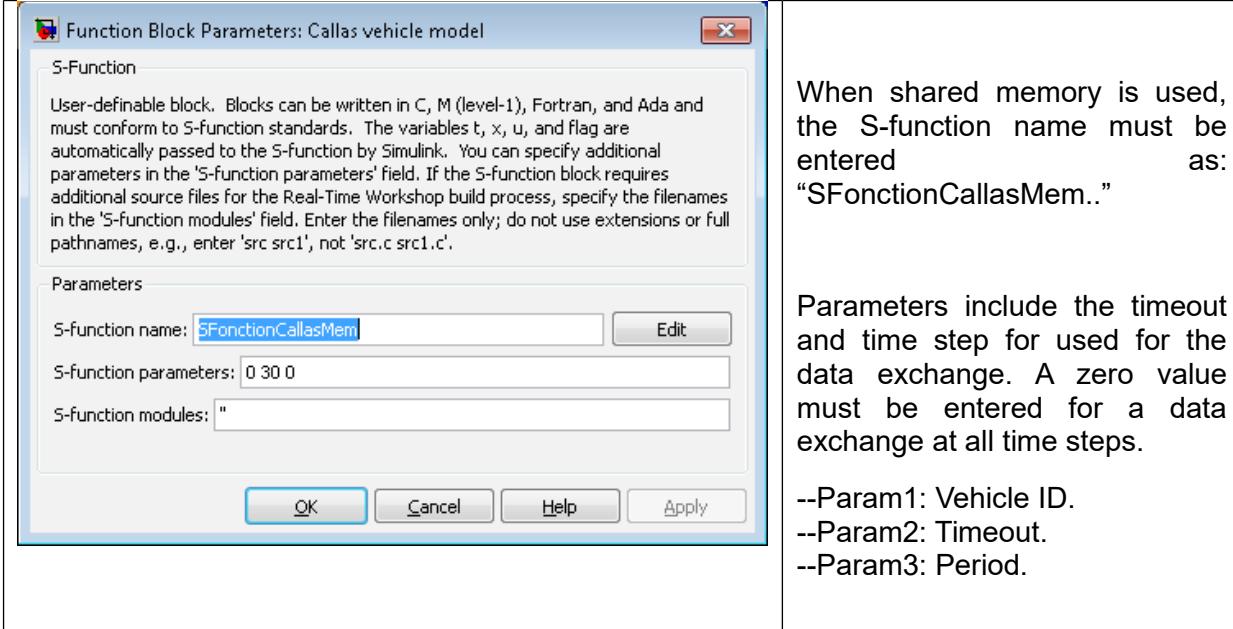
Illustration 76 : Browse network

5. The vehicle configuration is done, to continu refer to the next chapter Erreur : source de la référence non trouvée - Erreur : source de la référence non trouvée - Erreur : source de la référence non trouvée

X.E.4.iv.b - MatLab\Simulink settings

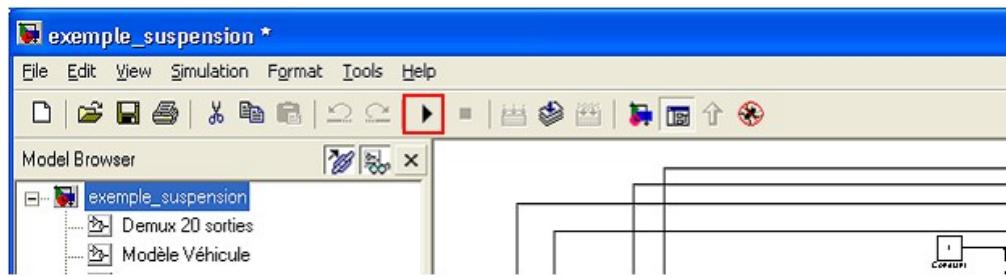
To select the communication type, the user will double-click on the “SCANeRstudio Vehicle model” block shown on the previous page.

The block properties are then displayed:



When a computation is launched in SCANeR™ studio, the modules are initialised the S-Function is recognised the number of Outputs set in the configuration file are displayed.

SCANeR™ studio then awaits Simulink's response. A computation must be launched in Simulink by clicking on the "Play" button.



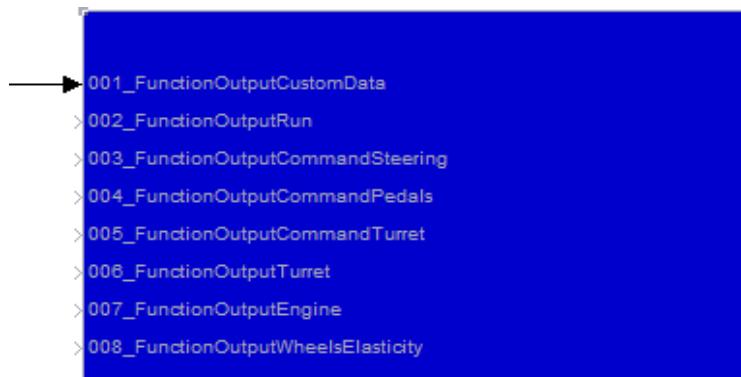
Synchronisation occurs and the simulation case is then computed within a co-simulation.

When the user wish to run a simulation again without modifying the SCANeR™ studio files (and only modifying the Simulink model), he will press on Simulink's "Play" button once again. The computation in SCANeR™ studio will automatically start.

We strongly recommend that the user run the first computation of a new session directly from SCANeR™ studio and not from MATLAB, as the files present in SCANeR™ studio working directory can have been modified by another user.

X.E.4.v - Display results in SCANeR™ studio

SCANeR™ studio channels can be accessed in the EG Browser (in SCANeR™ studio's graphical interface). When the user wish to store values from the Simulink model in SCANeR™ studio's results file, these values can be stored in the module custom output (1023 output per module). The first Vehicle Model block input is dedicated to custom module output.

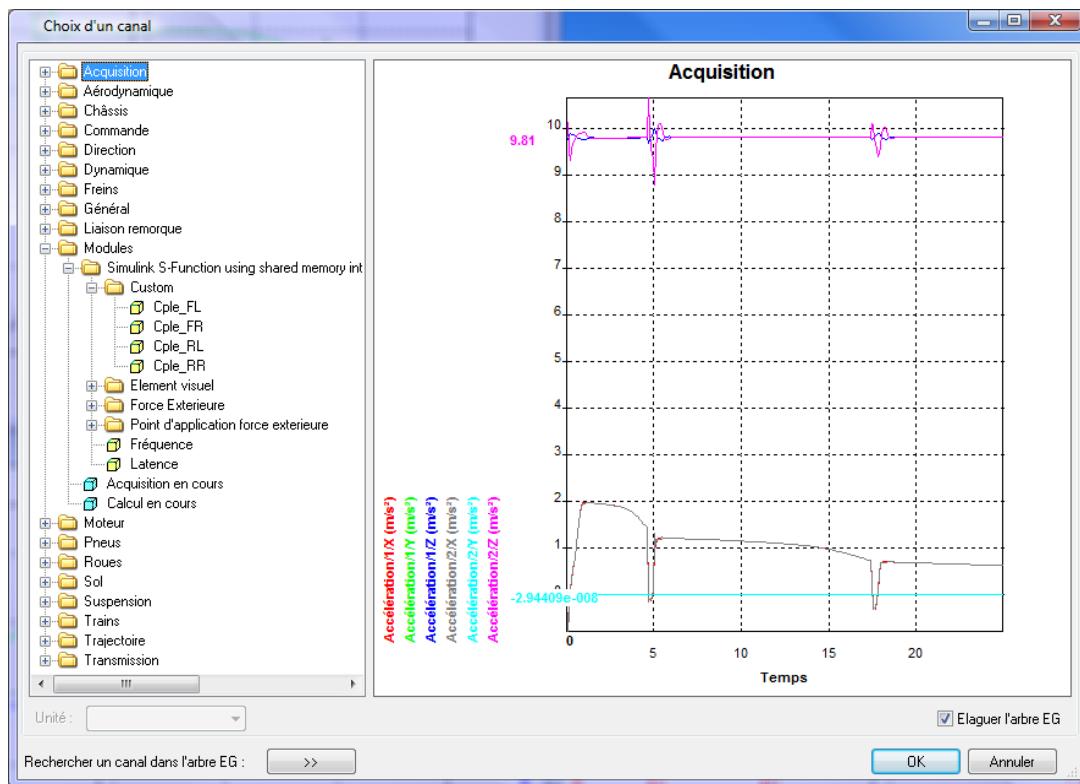


The user will remember to also connect the arrow inside the S-Function Block.

To use custom data, the user must:

- Add the block 001_FunctionOutputCustomData (5 entries but it can increase by adding simulink entries in this block)
- connect the variables to be stored to the new Mux block

These variables will become available in the EG Browser in **Modules/SimulinkS-Function/Custom:**



X.E.5 - VEHICLE MODEL BLOCK INPUTS

When creating a block diagram that communicates with SCANeR™ studio, the user must:

- Respect the size of the vectors (easy to check, as a co-simulation will not run when a mistake has been made)
- Respect the order of the variables transiting between Callas and Simulink.

X.E.5.i - Size of input functions

X.E.5.i.a - Generalities

A co-simulation offers numerous externalisation options through the Callas vehicle model block. In addition to the distinction made between internal models (which replace part of the vehicle model) and functional models (which add to SCANeR™ studio without replacing any of the model parts), we also need to differentiate single-component models from multi-component models:

- Single-component models only replace one part of a model. There can only be one such model per vehicle, such as: engine, aerodynamics, pilot, etc.
- Function inputs are in most cases single channels: each input port must be connected to a single arrow.
- Multi-component models replace part of a model that can be used several times on a vehicle (i.e.: suspensions, tyres, etc.). These models have a specific “IsExternalized” input whose size varies according to the selected function. They allow externalisation of certain vehicle components, such as suspensions for axle #1 for instance. Multi-component models can have 3 different sizes:
 - “for each axle”: 10

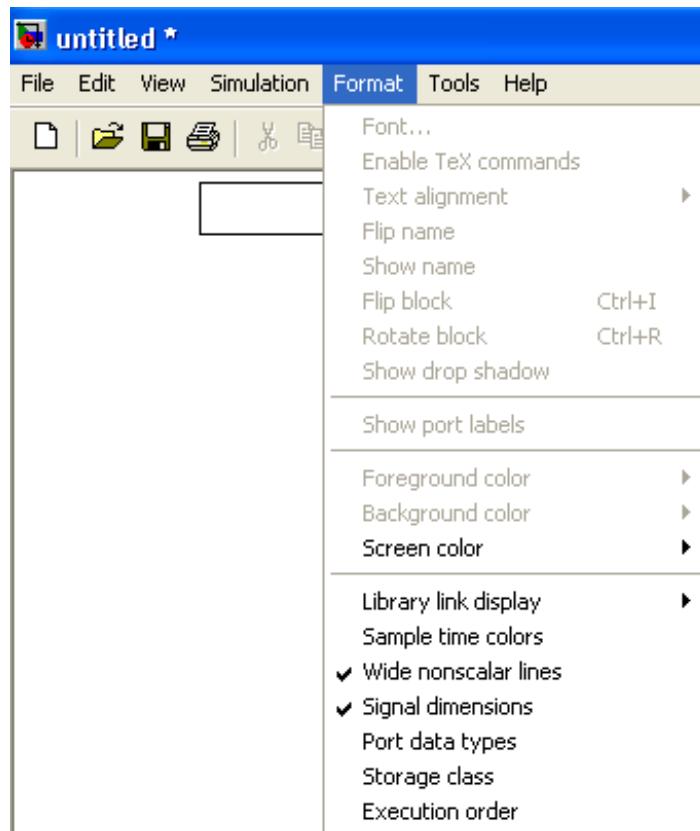
- “for each wheel”: 20
- “for each tyre”: 40

SCANeR™ studio has a required number of entries it must receive for each function. If, for instance, a modelled vehicle only has 4 tyres, the user is still required to send data relative to all 40 tyres potentially in use in the vehicle.

Multi-component model entries can have different sizes, making model design and use even more complex. Indeed, the distinction must be made between the “IsExternalized” data and the data that is proper to each function and which must be provided for each externalised component.

X.E.5.i.b - Recommended Simulink settings

To help with error diagnostic linked to dimension issues, we recommend that the user select both visualisation options displayed below:



“Wide non-scalar lines” plots a wide lines for vectors and thin lines for single values.

“Signal dimensions” is used to display the size of the vector transiting through the connection line. This dimension is updated when launching a computation.

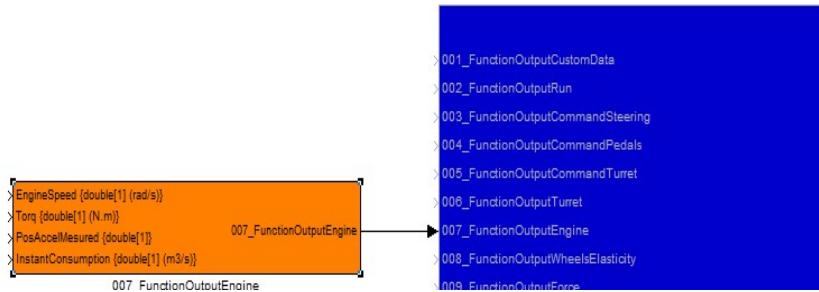
X.E.5.i.c - Examples

To clarify matters, let us examine the case of an engine, of tyre grip and of suspensions/elasticity in more detail in the following pages:

X.E.5.i.c.1 - Engine

- An engine is a single-component model
- There is only one engine per vehicle
- An engine has 4 inputs
- These inputs are scalar values. The size is specified on the function block

To externalise the engine model, the user needs only connect the 007_FunctionOutputEngine to the Vehicle model block entry port:



Each of the 4 007_FunctionOutputEngine entries is then connected to the part of the Simulink model that computes the corresponding variable.

X.E.5.i.c.2 - Tyre Grip

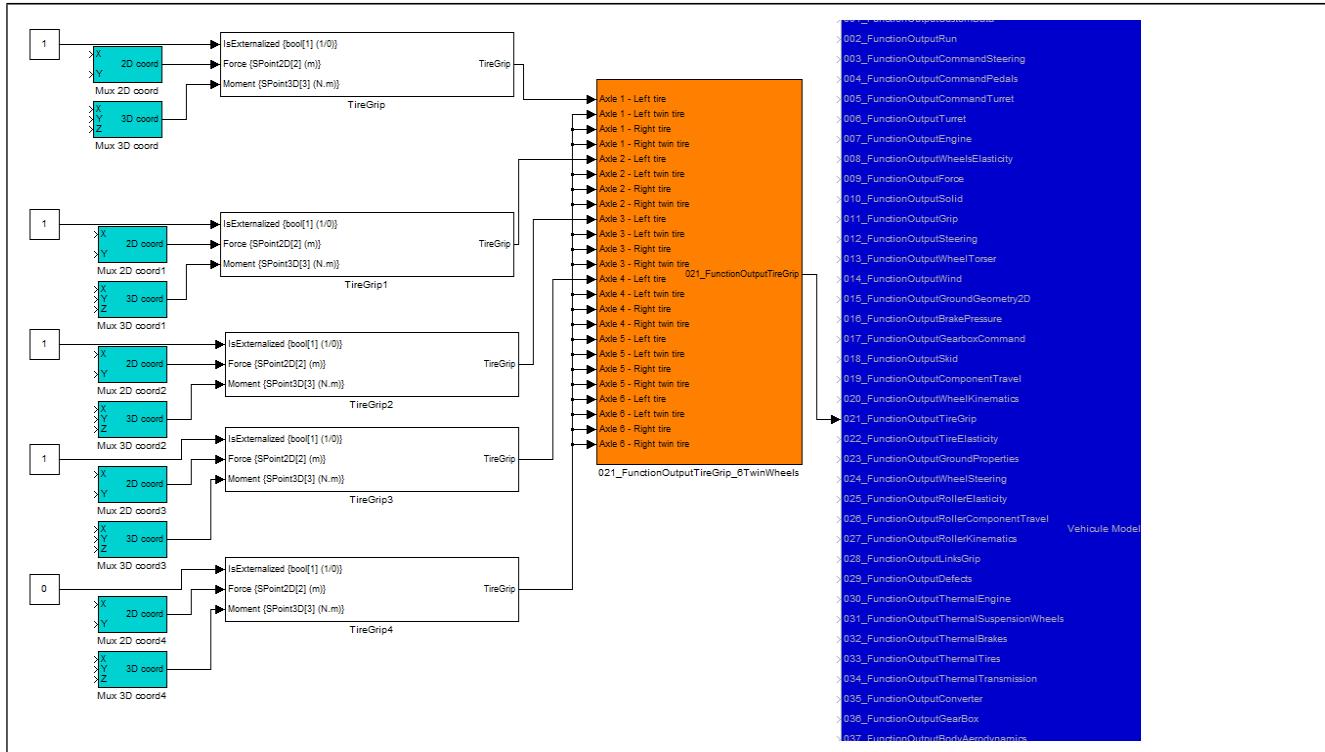
- Tyre grip is a multi-component model with same-size inputs

Externalising tyre grip requires more work. Let us assume we have a 4 axles vehicle with no twin wheels; we wish to externalise tyre grip for the left wheels only. Each tyre must indicate whether it is externalised and provide the data relative to the externalised function. A specific “TireGrip” block has thus been created, which regroups both the IsExternalized data and the data relating to tyre grip (including 2D force and 3D moment at contact point).

There are 3 input ports:

- “IsExternalized”: must be plugged to a constant value of 1 when the tyre is computed using Simulink, or a value of 0 when SCANeR™ studio’s tyre grip is used.
- “2DForce” must be plugged with a vector of dimension 2 for Fx and Fy (computed at the contact point location).
- “3DTorque” must be plugged to a vector of dimension 3 for Mx, My and Mz (computed at the contact point location).

<p>4 “TireGrip blocks must be added (from the SERA-CD Toolbox), one for each modeled tire grip.</p>	
<p>The “IsExternalized” input port must be connected to a constant value of 1, as these tyre grip values are computed in Simulink.</p>	
<p>The “2DForce” input port is connected to a Mux of size 2. The “3DTorque” input port is connected to a Mux of size 3.</p>	
<p>Each Mux input receives scalar values</p>	
<p>The 4 TireGrip blocks will be respectively connected at the following entries : Axle1-Left tire – Axle2-Left tire – Axle3-Left tire – Axle4-Left tire</p>	
<p>As it is, the connection is incomplete and 2 problems arise: SCANeR™ studio does not know whether the right wheels are externalised or not. A third Tyre block must therefore be created and connected to all non-externalised inputs. To tell SCANeR™ studio that the right tyres are not externalised, the “IsExternalized” input must be set to 0. The 2DForce and 3DMoments must still be connected to Muxs (so they have the proper dimension) and must be set to 0, even if these values will not be used.</p>	



The connection is now correct. Externalised tyres now have their own block that contains data only relating to them. All other blocks are connected to one block only containing zero values.

SCANeR™ studio will now know that MATLAB only manages 4 tyres located on the left side of the vehicle.

X.E.5.i.c.3 - Suspension – Elasticity

- Suspension elasticity is a multi-component model with different size inputs.

Externalisation of suspension is the most complex type of externalization, as the data used do not all have the same sizes. When each of a vehicle wheels is sprung, externalization will be managed at the axle. Both wheels of the same axle are always externalized. In this example, the vehicle will have two axles whose only the rear axles will be externalized. The "IsExternalized" input has a dimension of 10 while the "Elasticity" input has a dimension of 20. It is thus not the wheel that tells CALLAS whether it is externalised or not, but the axle it is attached to.

2 blocks have thus been created:

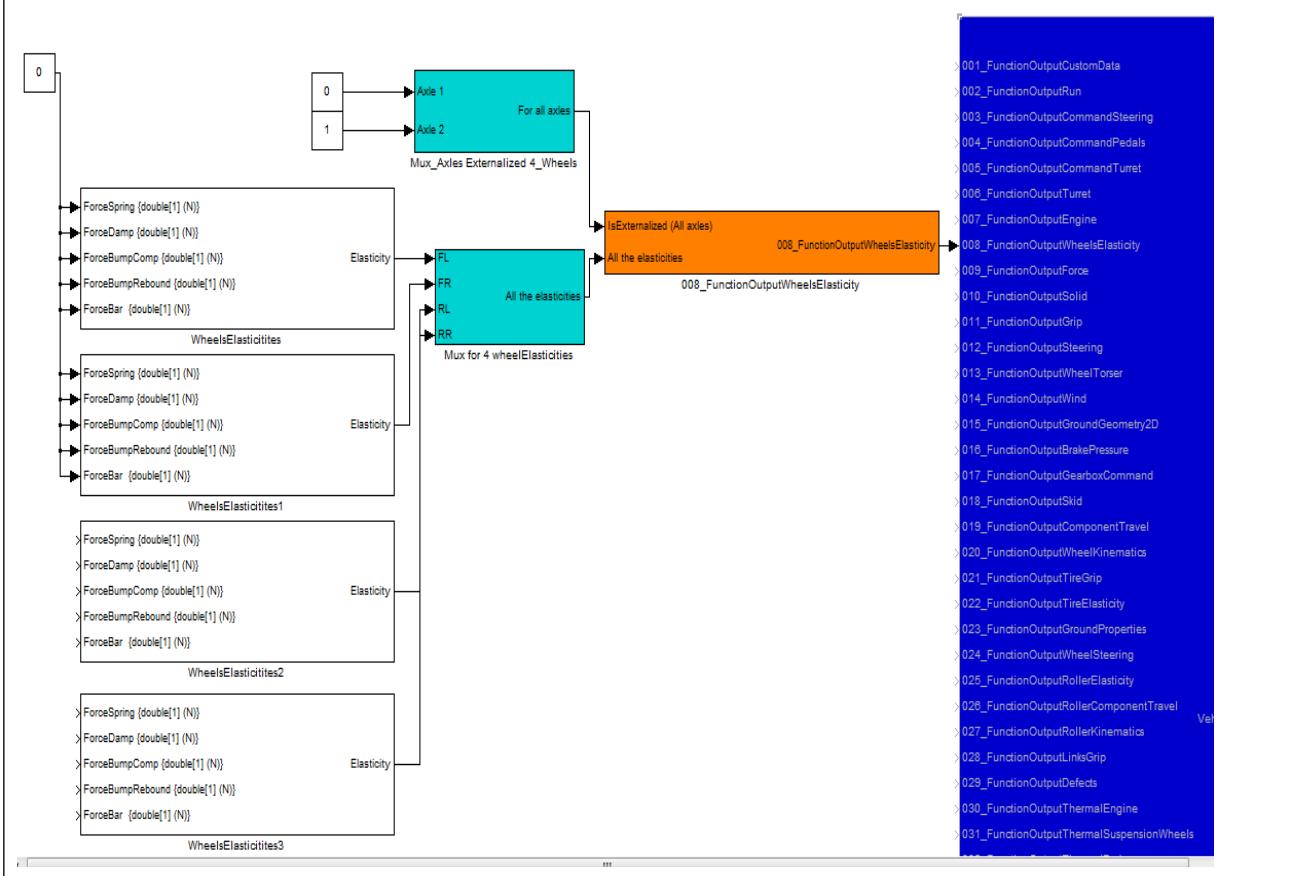
- A Mux_Axes_Externalized_4_Wheels to externalise 2 axles
- A WheelsElasticities block specific to the externalization of a suspension elasticity. This block contains the data relating to elasticity, including: spring force, damper force, bump stop force and anti-roll bar force.

<p>The “Mux_Axes Externalized 4_Wheels” Mux block tells SCANeR™ studio which axle/s are externalised and set to a constant value of 1. All other axles must be set to constant 0 values.</p>	
<p>As, in our example, each axle has 2 wheels, 2 “WheelsElasticities” blocks must be added using the OKTAL Toolbox.</p> <p>Each of these block inputs is connected to a model that computes the variables they contain.</p>	
<p>The “Is Externalized” input receives a vector of dimension 10.</p> <p>The “All elasticity” input is connected to a vector of dimension 100 (20 wheels, each containing 5 force values for the elements of one suspension)</p>	
<p>The two “WheelsElasticity” blocks must now correctly be plugged into the Mux for 4WheelsElasticities.</p> <p>A Mux of dimension 20 must be used.</p>	
<p>A third block called “WheelsElasticity”, full of constant 0 values, is plugged into the remaining inputs ux for 4WheelsElasticities.</p>	

Lastly, the following connections must be made:

- The “Mux_Axes Externalized 4 Wheels” output must be plugged into the “IsExternalised” entry port (in the Suspension - Elasticity Mux block).
- The “Mux for 4WheelsElasticities” Mux output must be plugged into the “All elasticities” entry port (in the Suspension - Elasticity Mux block).

The diagram is now correct. All data transiting between the various blocks have the correct sizes.

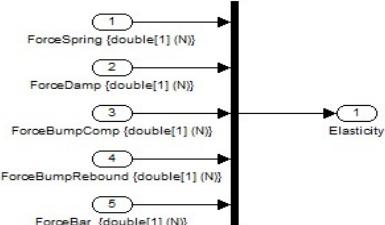


X.E.5.ii - Order of inputs and outputs

We have just seen that the size of data transiting between blocks is important. The transit order of the data transit in is also crucial for a co-simulation to function properly.

SCANeR™ studio sends and receives values without knowing which variables they relate to. It is the order in which these variables are sent and received that helps to define them.

This helps to better understand the advantage of using pre-set blocks with clearly labelled inputs and outputs to avoid errors.

<p>In our previous example, we worked with the following "WheelsElasticities" block:</p>	 <p>The block content consists of five sub-blocks: ForceSpring {double[1] (N)}, ForceDamp {double[1] (N)}, ForceBumpComp {double[1] (N)}, ForceBumpRebound {double[1] (N)}, and ForceBar {double[1] (N)}. These are grouped under the label "Elasticity".</p>
<p>When looking at the block content, we can see that it only contains a Mux of dimension 5, with inputs labelled to define the order in which they must be sent to SCANeR™ studio.</p>	 <p>The internal structure of the block shows a Mux block with five inputs labeled 1 through 5. Each input is connected to one of the five sub-blocks: ForceSpring, ForceDamp, ForceBumpComp, ForceBumpRebound, and ForceBar. The output of the Mux is labeled "Elasticity".</p>

<p>We have also worked with the “Mux for 4 WheelsElasticities” block: to connect the WheelsElasticities in the input block function : all the elasticity</p>	<pre> graph LR FL[FL] --> Mux[Mux for 4 wheelElasticities] FR[FR] --> Mux RL[RL] --> Mux RR[RR] --> Mux Mux -- "All the elasticities" --> Out[] </pre>
<p>This block only contains a Mux of dimension 20.</p> <p>The first four inputs are defined as the first four inputs of the Mux.</p> <p>All other inputs are connected to a Wheels elasticities block with only 0 values to complete SCANeR™ studio dimension.</p> <p>This is a preplanned block for a 4 wheels vehicle.</p>	<pre> graph LR Input[0] --> WE[WheelsElasticites] WE -- "ForceSpring [double[1][N]]" --> Elasticity[Elasticity] WE -- "ForceDamp [double[1][N]]" --> Elasticity WE -- "ForceBumpComp [double[1][N]]" --> Elasticity WE -- "ForceBumpRebound [double[1][N]]" --> Elasticity WE -- "ForceBar [double[1][N]]" --> Elasticity Elasticity --> Stack[] Stack --> Out[] </pre>

Inputs must be sent to SCANeR™ studio in the following order:

- For axle tables: from axle 1 to axle 10

- For wheel tables:

- Axle 1, left wheel
- Axle 1, right wheel
- Axle 2, left wheel
- Axle 2, right wheel
-
- Axle 10, left wheel
- Axle 10, right wheel

- For tyre tables :

- Axle 1, left tyre
- Axle 1, left twin tyre
- Axle 1, right tyre
- Axle 1, right twin tyre
- Axle 2, left tyre
-
- Axle 10, right twin tyre

- For tank rollers (including pulley and sprocket):

- Roller 1- left
- Roller 1- right
- Roller 2- left
- Roller 2- right
-
- Roller 9- right

For all other functions, the order of inputs will be defined by the supplied blocks or provided in the relevant documentation: "Exhaustive list of externalised functions".

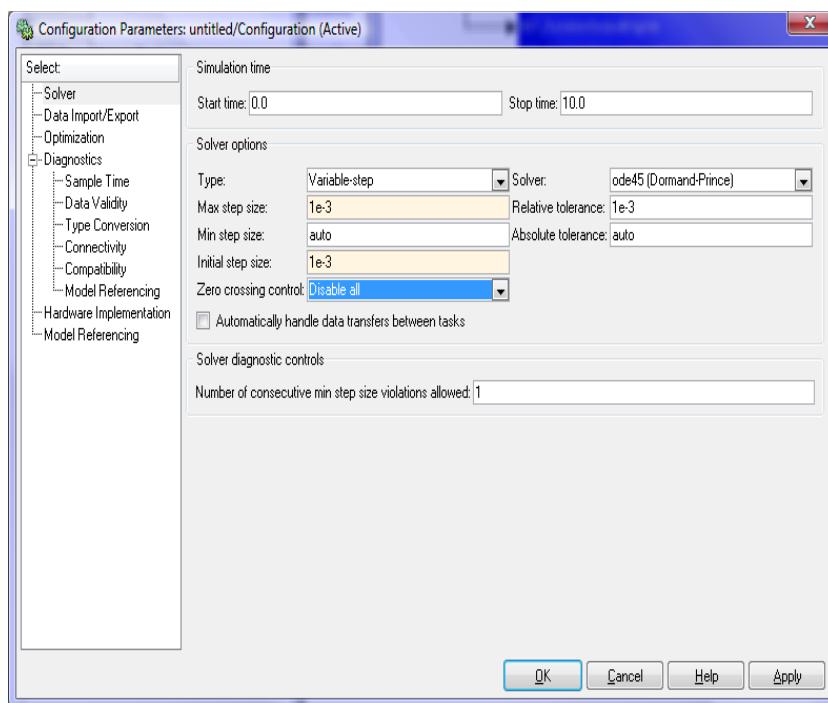
X.E.5.iii - Setting up a simulation in MatLab/Simulink

X.E.5.iii.a - Numerical method

MATLAB/Simulink offers a number of numerical methods. The method used will depend on the modelled system stiffness and content. Whichever method is selected, the user must:

- Define the end time
- **Imperatively set the maximum step size.** When no step size has been defined, Simulink may use a large time step that is compliant with its model, regardless of what happens in SCANeR™ studio and leading to divergences in the Vehicle Model. A 0.001s maximum limit for a 4-wheel vehicle is a good compromise between speed, robustness and quality of results.

This is the setting for Matlab 7:



The integration method is only valid for Simulink, as SCANeR™ studio will use the set-up defined in the "UTI" file. The user will then need to select between "Single step" (Euler) and "multiple steps" (RK4). Variable step and real-time are not valid choices (only possible with a TCP/IP communication type).

X.E.5.iii.b - Diagnostic options

In Simulink's "Diagnostic" tab, the user can define the diagram auto check.

Consistency checking verifies the continuity of variables transiting through certain block types (such as: integrator, derivative, etc.).

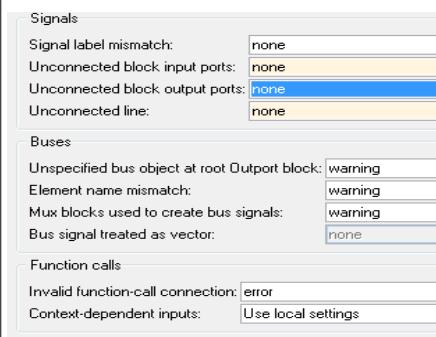
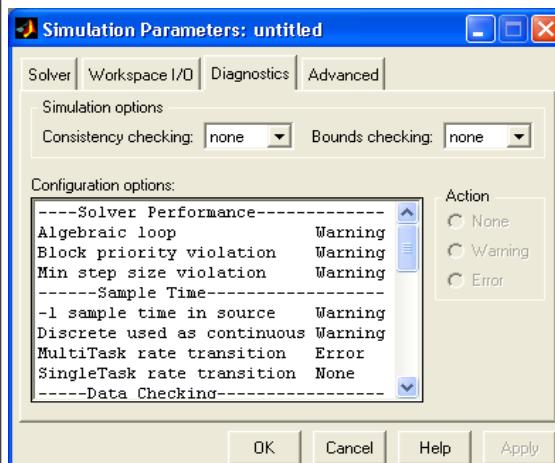
Bounds checking controls memory management.

Both of these options can be disabled.

The Warning option can also be enabled, but please note that it will slow a simulation.

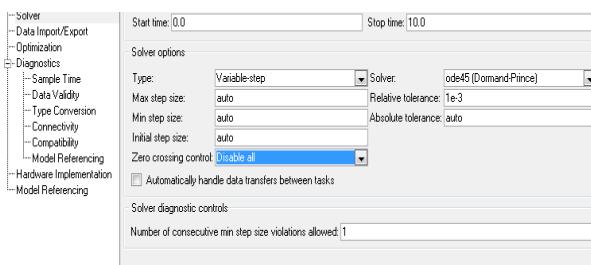
As the SCANeR™ studio S-Function must have disconnected blocks, numerous error messages in MATLAB's command screens will appear.

Warning messages can be avoided by setting the *Unconnected block input*, *Unconnected block output* and *Unconnected line* to **None**.



X.E.5.iii.c - Advanced settings

In the “Advanced” tab, the user can set potential MATLAB optimisations during the execution of a block diagram. This is the setting for Matlab 7:



When none of the blocks used in a diagram performs any particular operation when getting to zero values, disabling “Zero crossing detection” can improve upon simulation duration time. Please refer to the relevant Simulink chapter for further details.

X.F - CALLAS SIMULINK SUB-MODEL (RTW)

X.F.1 - OVERVIEW

The Callas Simulink ERT target is used to integrate custom modules in SCANeR studio Callas vehicle model.

This interface should be used to develop models of vehicle sub-systems such as ABS brakes, engines, transmissions, active suspensions, ADAS etc. These subsystems can then be used with a Callas vehicle model.

It is not possible to use this interface to integrate a chassis model or a complete vehicle model. To integrate a complete vehicle model in SCANeR studio, use the “VehicleDynamics” API, documented separately.

X.F.2 - GENERAL CONCEPT

The vehicle sub-system developed with the laces parts of the vehicle model.

Each sub-system is defined by the list of inputs it has, and the functions of the Callas vehicle that it replaces.

The inputs of the sub-system can include any of the outputs that are computed by the Callas vehicle model. Such as vehicle speed, position, engine speed, wheel speed etc. See input/output chapter for details.

The functions that the sub-system replaces defines what action the sub-system will have on the vehicle. These functions include steering, engine, brake pressure etc. see input/output chapter for details. Each function of the sub-system that is active will disable the native Callas model for the given function. For example, if the sub-system has the steering function, the Callas model steering function will be replaced.

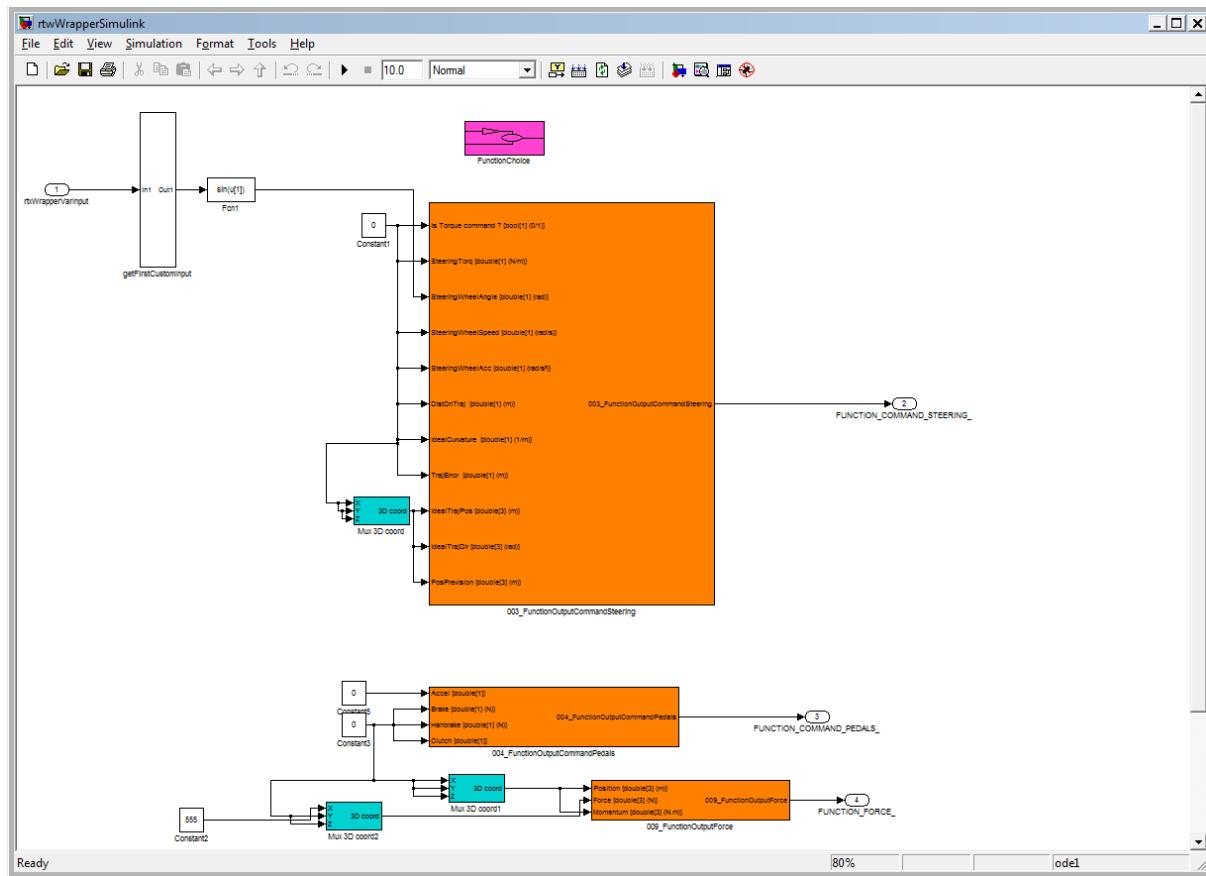
X.F.3 - BUILDING A SIMULINK VEHICLE SUB-SYSTEM SAMPLE

Open the sample Simulink vehicle sub-system model located in:

<STUDIO_PATH>\<STUDIO_VERS>\APIs\samples\VehicleDynamicsCallas\RTWWrapper\rtwWrapperSimulink.mdl

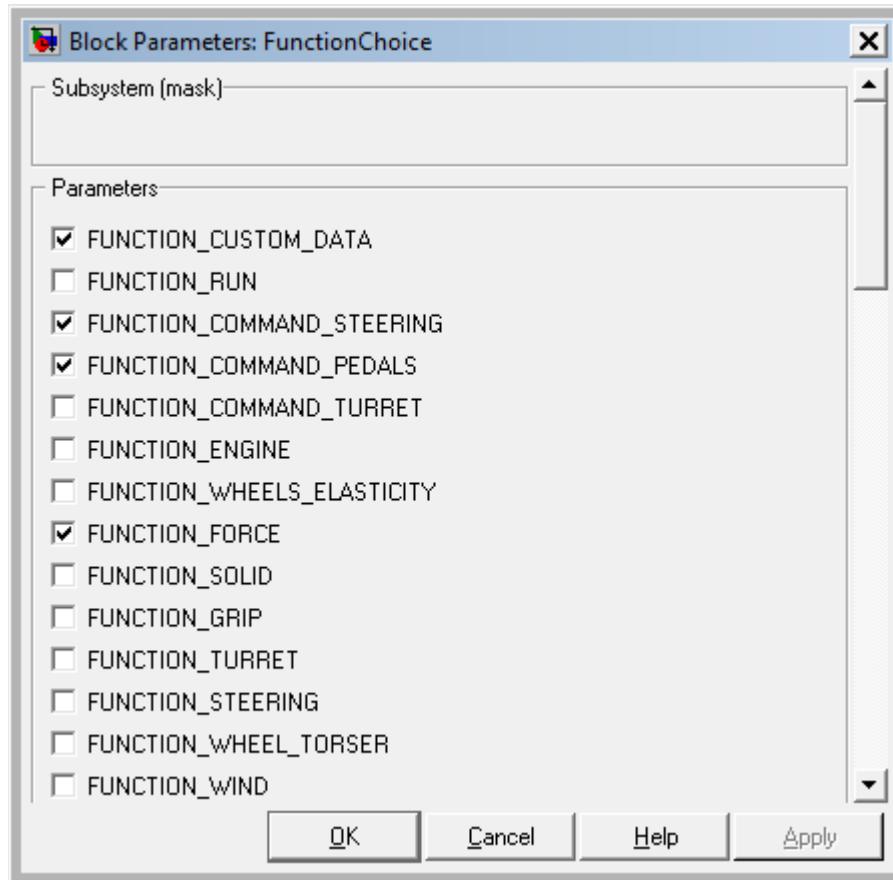
click “tools->real time workshop->build model”

This will build the project and produce a dll containing the vehicle sub-system model. See “external models” chapter on how to use an external model.

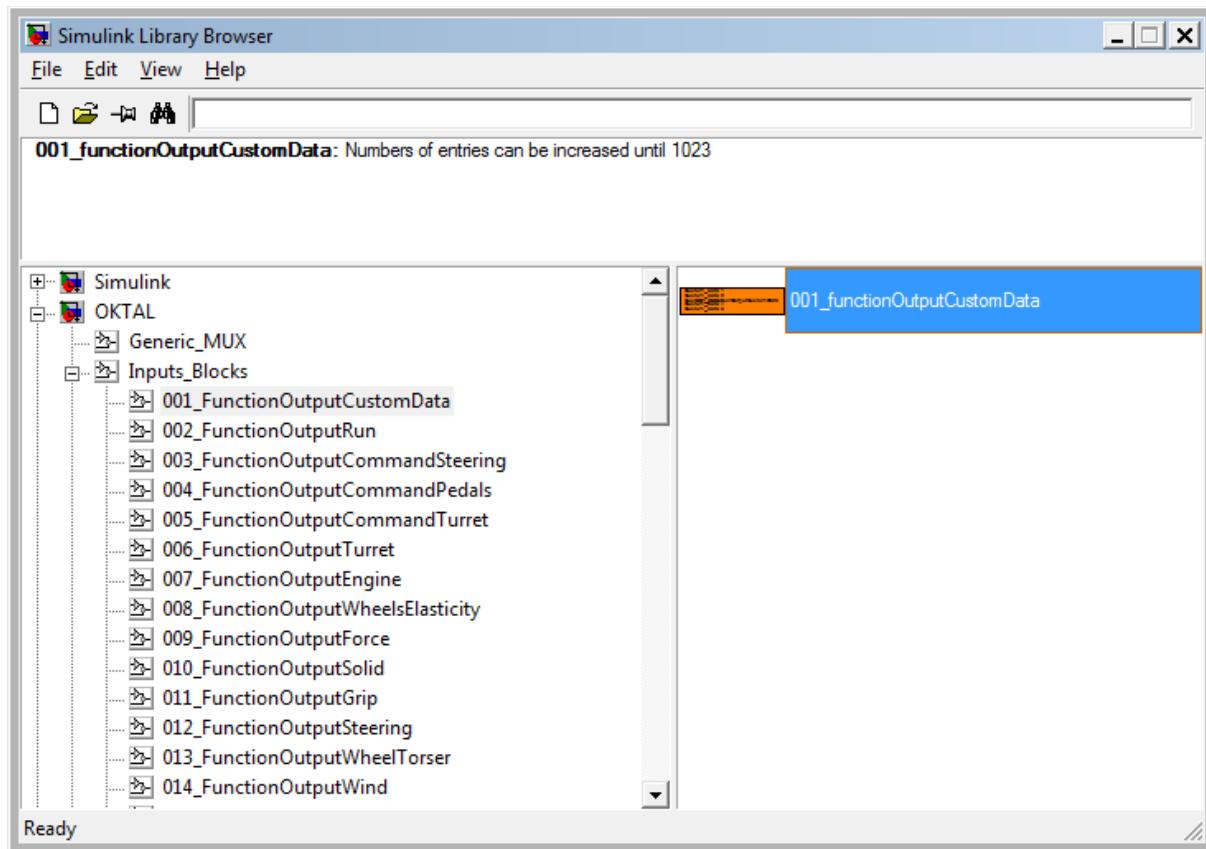


X.F.4 - CREATING A CUSTOM VEHICLE SUB-SYSTEM IN SIMULINK

The functions of the vehicle that can be overridden can be selected using the “FunctionChoice” Simulink block.



For each function checked in the FunctionChoice dialog, an output port appears in the Simulink model. Also, for each function there is an associated Mux with named ports in the Oktal/Input_Blocks section of the library browser.



#Input From CALLAS

131072, //Vehicle1/Generalities/Time (s)
 262144, //Vehicle1/Dynamic/Travelled distance (m)
 393216, //Vehicle1/Generalities/Spacing (s)
 524288, //Vehicle1/Dynamic/Slip angle (rad)
 655360, //Vehicle1/Dynamic/CoG position/X (m)

#Output From model to customData

Test1 (ms)

Test2 (°)

Test3 (km)

X.G - USE AN EXTERNAL MODEL ON A VEHICLE DYNAMICS CALLAS

An external model can be configured for use in two different locations:

- In the vehicle element (or organ) dialog box. A validation system enables the user to test the compatibility between his model and the vehicle element
- In the element interface.

The efficiency of an external model will depend on the way it was developed. A number of features may or may not be available to the user as a result. Some of these features enable the user to:

- Save the external model's specific data in a separate configuration file
- Integrate this configuration file into SCANeR™ studio vehicle file
- Edit the configuration file through a graphic interface
- In the case of multiple components, the user must specify all functions that are externalized for each element organization



Please note that any malfunction in the external model will cause for SCANeR™ studio to malfunction.

We recommend that the user of an external model refer to the appropriate "Developer's documentation" to gain familiarity with the model mechanisms and functions.

X.G.1 - PRESENTATION

When a vehicle uses no external models, an element Edit box will appears as follows:

- The "External model" box is unchecked
- The external model parameters are disabled (greyed-out)

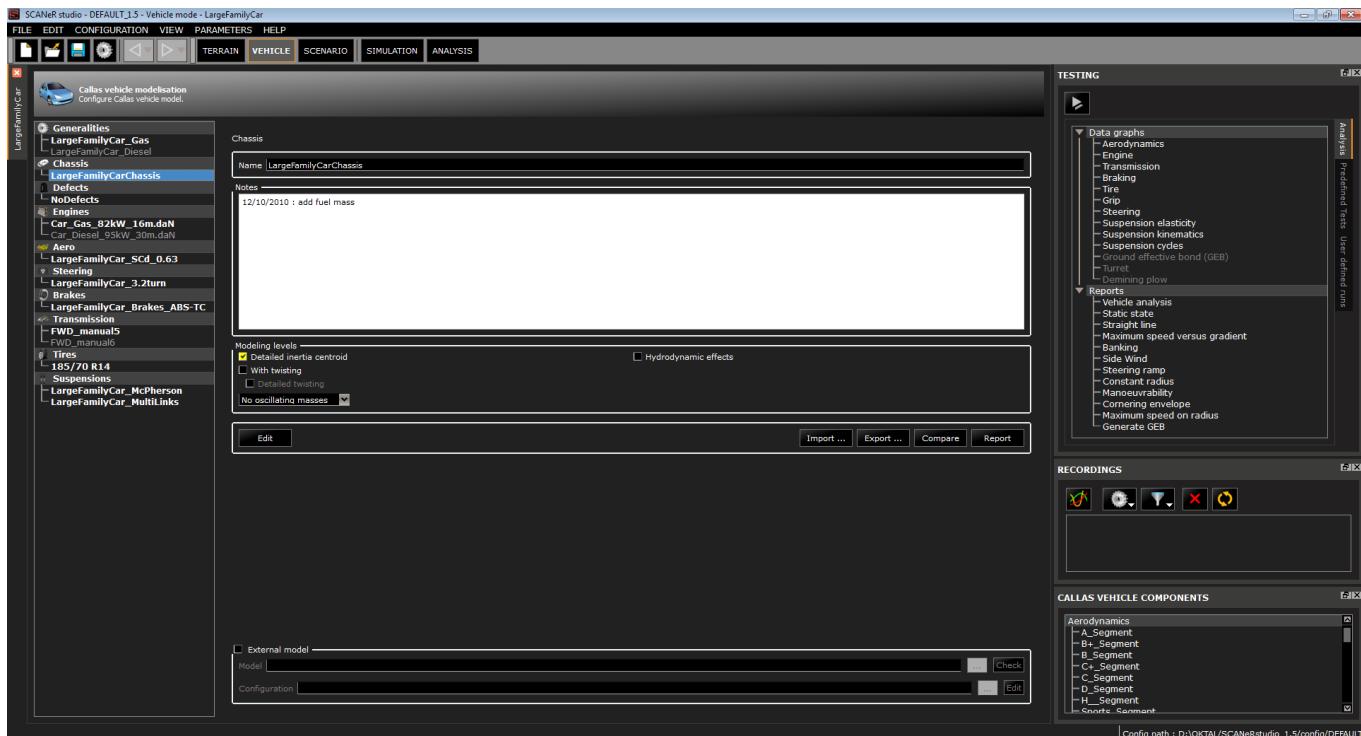


Illustration 77 : SDK – No external model used

To specify an external model, the user will start by ticking the “External



Illustration 78 : SDK – External model used

To select an external model, click on the “...” button on the model line.

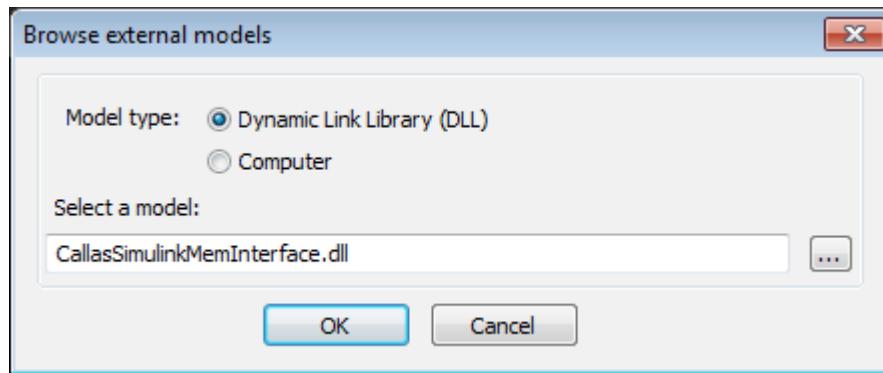


Illustration 79 : External model selection



When loading an external model or when changing models, the name of the configuration file is automatically reset.



When an external model is under the folder <STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>\plugins\Callas, only the name of the dll is needed, otherwise user needs to provide the absolute path.



The vehicle mode platform needs to be the same as the external model, otherwise you will get a compatibility error. Use vehicle mode in 32 bits with a 32 bits external model and vehicle mode in 64 bits with a 64 bits external model. For more details refer to chapter "**x64**" tab into the **SCANeRstudio_VEHICLE** documentation.

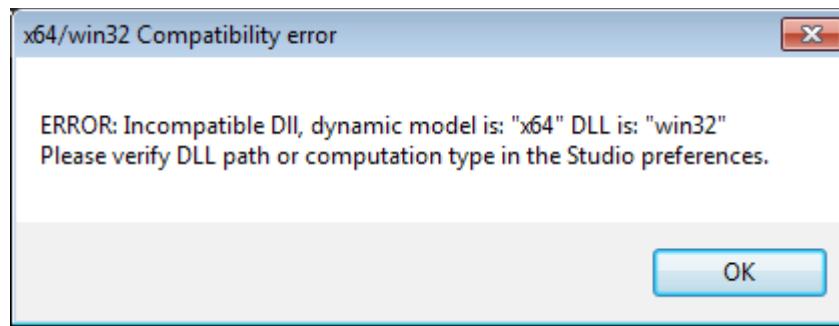


Illustration 80 : Compatibility error

If the model developer has specified a configuration file, the user must select a file.

Once the external model and its configuration file have been selected, the user will click on the “Check” button to validate the settings.

X.G.1.i - Checking

This function tests whether the vehicle element is compatible with the model's exported functions. It also checks that the DLL and configuration file paths are still valid.

This function is enabled by clicking on the “Check” button. The following dialog box is then displayed:

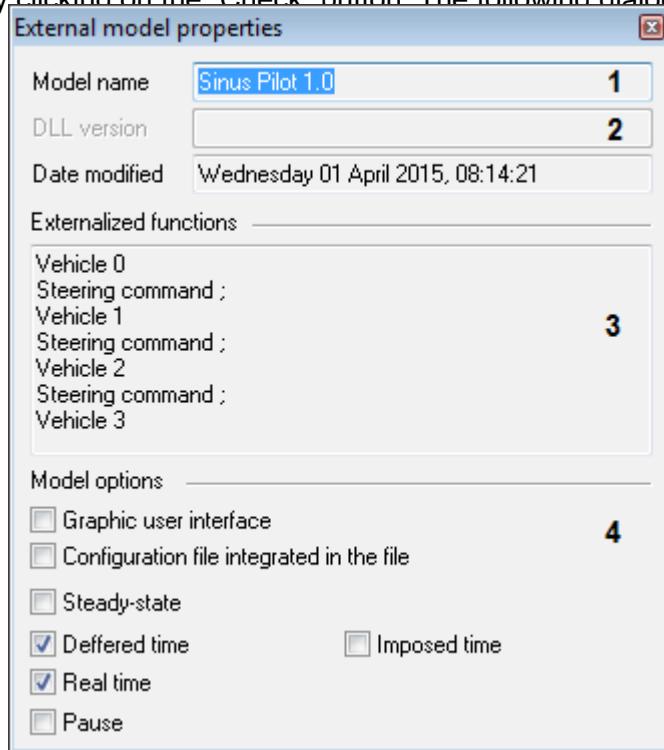


Illustration 81 : External model checking

The external model name (1) is displayed at the top of the dialog box and must not be mistaken with the dll name. The dll version number and version type (RELEASE or DEBUG) are indicated below the model name. (2).



Please note that this data is optional and that is not required in the external model.
When not implemented, all above boxes are disabled.

X.G.1.i.a - Externalized Functions

The list of compatible functions supported by the external model (3) is displayed below the dll version number. An incompatible external model is a model that exports a number of functions that do not match the vehicle component the model is loaded into.



User has the responsibility to load an external model on a compatible component.

X.G.1.i.b - Model Options

The model options (4) are displayed in the bottom part of the box. Only options supported by the external module are ticked.

Presence of a graphic user interface:

A user interface has been designed by the model developer. In this case, the button dedicated to the external model is enabled.

Configuration file integrated into the vehicle file:

Indicates that the configuration file is saved in the vehicle file. When this is the case, there is no need for a separate configuration file. All options are stored in the vehicle file and all modifications can be cancelled using the “Cancel” button.

Steady-state:

The external model can be used in steady-state computations.

Time-lag:

The external module can be used in time-lag simulations.

Imposed time:

The model imposes its own time scale. When using the Matlab/Simulink function, the Simulink time scale is used.

Real time:

The external module can be used in “real time” simulations, such as in a real time driving simulator.

Pause:

The external model supports a “pause” mode.



A number of messages can be displayed during the configuration stages of an external model. For more information on these messages and the dialog boxes they originate from, please refer to the external model's documentation.

X.H - GLOSSARY

Word or expression	Explanation
Block	Name given to Simulink library members
Co-simulation	Shared computation between 2 software programmes, each one computing part of the entire model.
External model	An external model replaces or completes part of SCANeR™ studio 's Vehicle model.
Functional model	Model that adds an extra layer to the SCANeR™ studio model, without replacing any of its parts.

Word or expression	Explanation
Multi component model	Model used by several parts of a vehicle (suspensions, tyres, etc.).
Internal model	Model that replaces part of the SCANeR™ studio Vehicle model.
Single component model	Model only used once in a vehicle (i.e: engine)
S-Function	Specific Simulink block; enables the programming of functionalities in C or C++. Handles communication between SCANeR™ studio and Simulink.
Toolbox	Generic name for Simulink libraries

X.I - ANNEXE 1: VEHICLE DYNAMICS CALLAS SAMPLES IN SCANeR™ STUDIO ENVIRONMENT

X.I.1 - EXTERNAL DEFECTS SAMPLE

X.I.2 - MATLAB EXTERNAL DEFECTS SAMPLE

X.I.3 - SIMPLE ENGINE SAMPLE

Model for an engine. This sample demonstrates how to implement a user interface within a module.

X.I.4 - MATLAB SIMPLE ENGINE SAMPLE

Model for an engine. This sample demonstrates how to implement an external engine using Matlab/Simulink in Co-simulation.

Here is the principle of externalizing the powertrain unit:

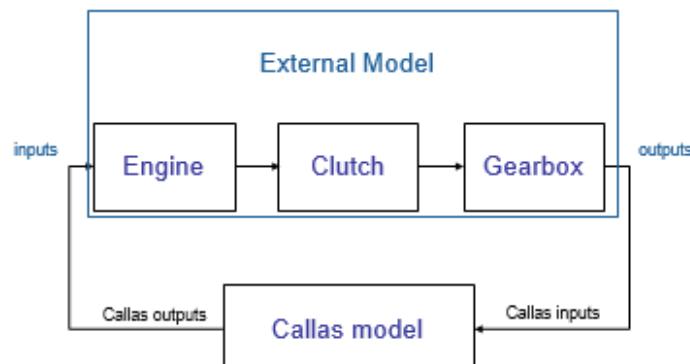


Illustration 82 : Externalization idea

In SCANeR, you need to think different and externalize each part like following:

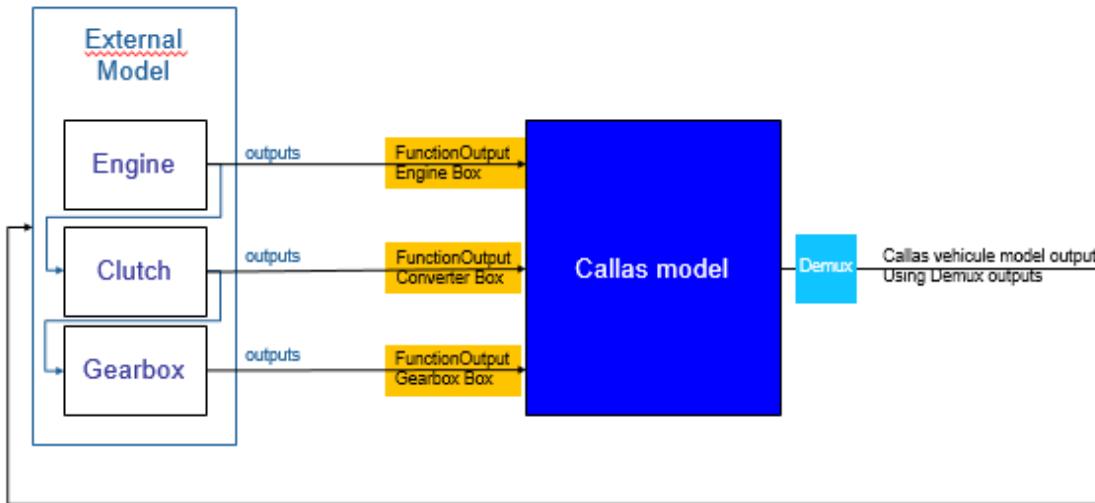


Illustration 83 : Externalization in SCANeR API

That means if you want to externalize all powertrain unit, you need to replace each part with one block like in parallel even if in reality they are in series! You cannot externalize everything in one time, you need to adapt your model and give to Callas what it needs to run.

The sample gave by OKTAL externalize only the engine.

X.I.5 - MATLAB SIMPLE ENGINE RTWWRAPPER SAMPLE

Model for an engine. This sample demonstrates how to implement an external engine using Matlab/Simulink with rtwWrapper. You can validate your external model with Co-simulation (MatlabSimpleEngine) and then build a dll and implement it in your Callas vehicle model (MatlabSimpleEngine_rtwWrapper).

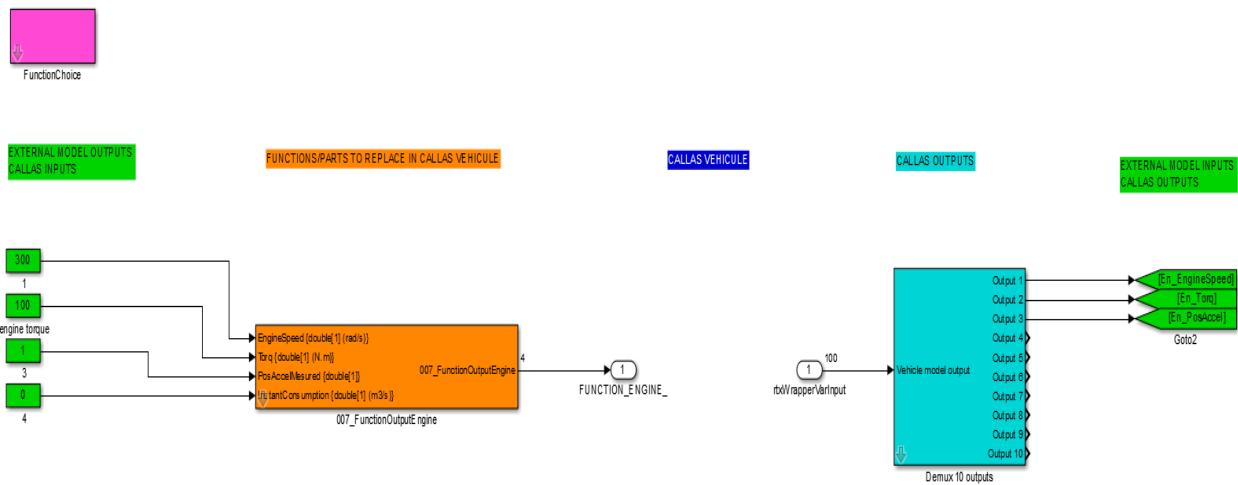


Illustration 84 : Simulink model

You can copy/paste what you have validated in Co-simulation. The principle is exactly the same, only the method of running is different.

Following, you can find rtwWrapper vs Co-simulation design differences:

rtwWrapper

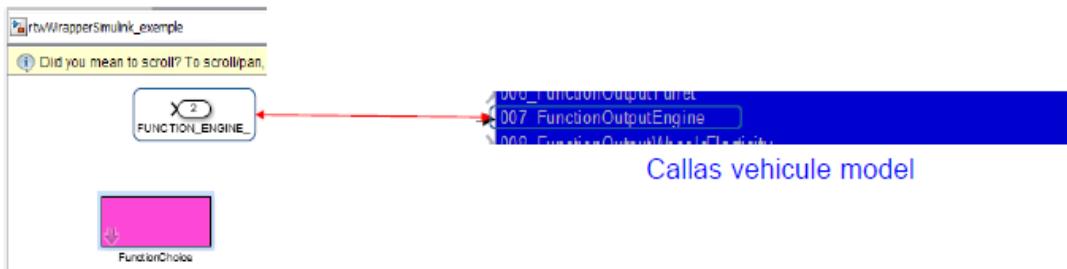
Equivalence in simulink for Cosimulation

Illustration 85 : rtwWrapper vs Co-simulation design differences

rtwWrapper

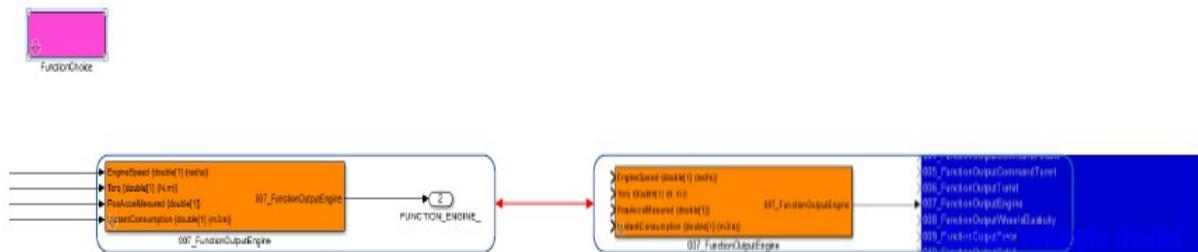
Equivalence in simulink for Cosimulation

Illustration 86 : rtwWrapper vs Co-simulation design differences

rtwWrapper

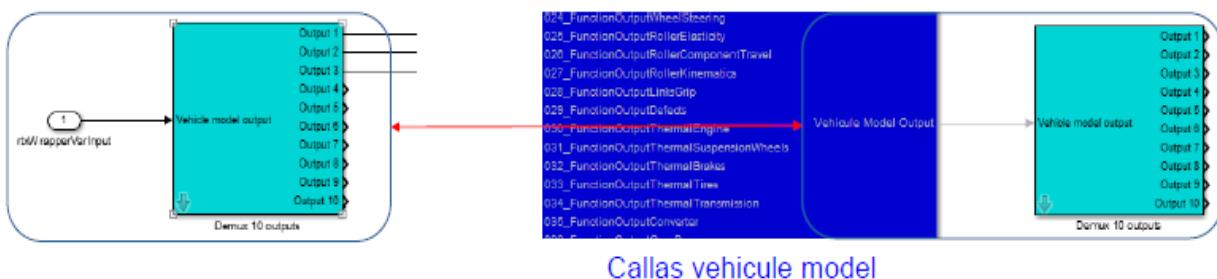
Equivalence in simulink for Cosimulation

Illustration 87 : rtwWrapper vs Co-simulation design differences



On the sample, you have a constant inputs for the engine.

X.I.6 - SINUSPILOT SAMPLE

Model that sends a sinus wave to the vehicle steering model.

X.I.7 - MATLAB SINUSPILOT SAMPLE

X.I.8 - SINUSPILOT_C

Equivalent to SinusPilot for those more comfortable with C.

XI - VEHICLE DYNAMICS API

The VehicleDynamics API is the SCANeR™ studio interface to integrate External vehicle models in SCANeR™ studio. This is to integrate a complete vehicle model, including simulation of the chassis, steering, engine, suspension, tires and aerodynamic. The model to be integrated must simulate all these parts of the model.

If the model only included one part of the model, it can be integrated in the SCANeR™ studio Advanced model using the VehicleDynamicsAdvanced API.

The API is in C++, but there is a Simulink binding to integrate Simulink models.



Currently SCANeR™ studio is compliant with 32bits. Read more in the "Environment" chapter (page 18).

XI.A - BUILDING A VEHICLE DYNAMIC SAMPLE

XI.A.1 - PACKAGE

This package includes:

- A DLL file: **VehicleDynamicsSample.dll** (located in **<STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013\plugins**) & .
- An include file, **SampleEditableVehicle.h** (located in **<STUDIO_PATH>\<STUDIO_VERS>\APIs\samples\VehicleDynamics\VehicleEditionSample**)
- This documentation
- A MS Visual C++ 12 solution which contains some examples written in C++ (**<STUDIO_PATH>\<STUDIO_VERS>\APIs\VehicleDynamics\samples\VehicleDynamicsSample\VehicleDynamicsSample.vcproj**)
- The delivered "VehicleDynamicsSample" model use the **VehicleDynamicsSample.dll**. (read more about "External model" in the dedicated chapter of the SCANeRstudio_VEHICLE_UserManual documentation and check the "DEFAULT\Studio_ExternalBasedOnVehicleDynamicSample.sce" scenario in the "DEFAULT scenarii" chapter of the SCANeRstudio_VEHICLE_UserManual documentation).

XI.A.2 - C++ BUILD ENVIRONMENT

Visual Studio 2013 must be installed on the PC used to compile a C++ vehicle model.

Open the C++ vehicle model

Select the "Release" build configuration.

XI.A.3 - BASE FACTORY INTERFACE

The base interface for the DLL is one C functions for creating a C++ implementations of IVehicleDynamics objects.

ScanerPluginInterface.cpp

```
#include "VehicleDynamicsCustom.h"

extern "C" __declspec(dllexport) cats::model::IVehicleDynamics * createVehicleDynamics()
{
    return new CustomModel();
}
```

XI.A.4 - BEHAVIOUR DESCRIPTION

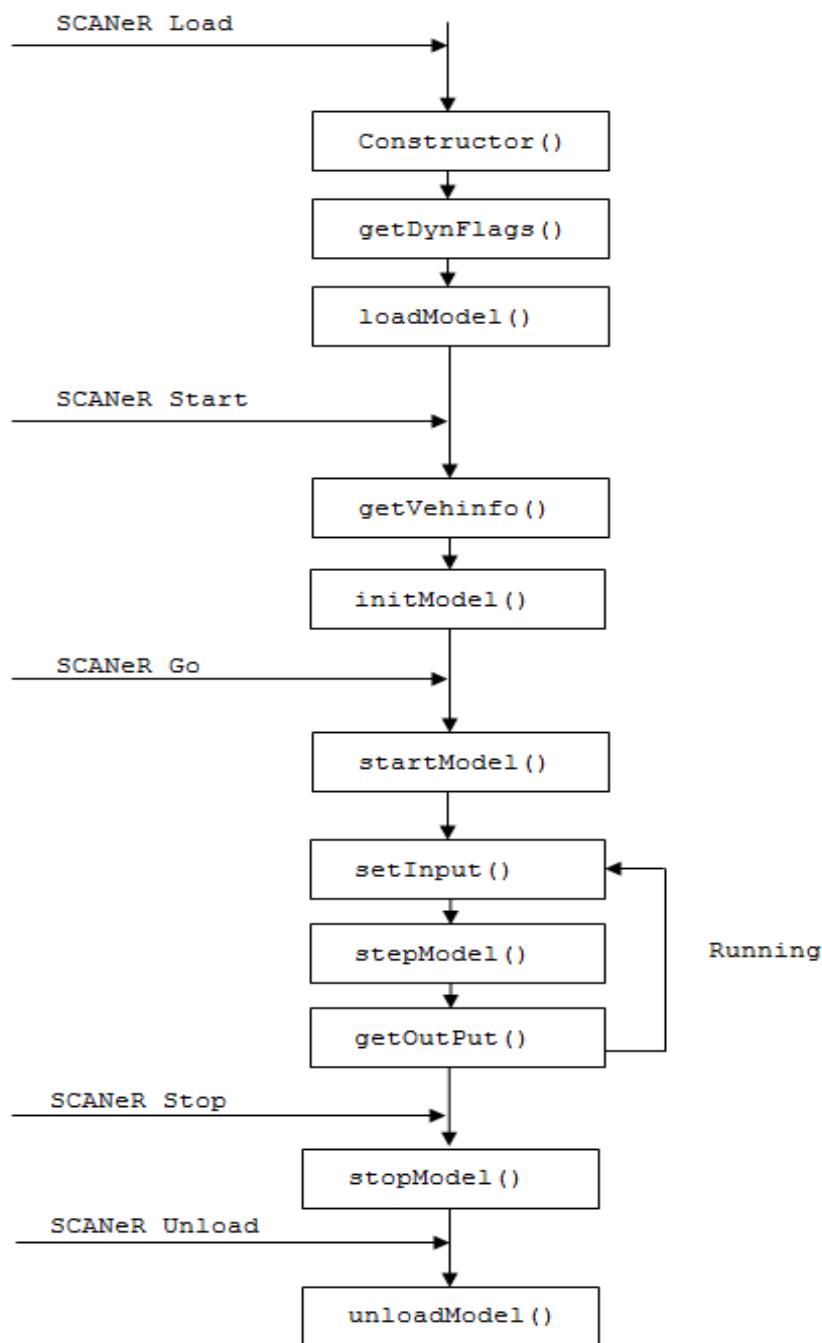
The implementation of the dynamic model:

1. is done as show below (XI.A.4.i - "Model implementation" - page 193)
 2. manages a set of custom Input/Ouput:
 1. custom Input 0 is used to compute the custom Output 0 (= 2 x custom Input 0)
 2. covered distance is used to set the custom Output 1
- Others are not implemented

XI.A.4.i - Model implementation

The implementation of the dynamic model is done by sub classing the **cats::model::IVehicleDynamics** abstract interface. The interface is describe in the delivered header files:

- **IVehicleDynamics.h**: main interface.
- **VehicleInput.h**: vehicle inputs data structures.
- **VehicleOutput.h**: vehicle outputs data structures.
- **WheelState.h**: vehicle wheels output data structures.

Illustration 88: *cats::model::IVehicleDynamics*

XI.A.4.ii - IVehicleDynamics

```

//! This abstract base class defines the common interface of all vehicle dynamic models.
/*! A plugin should implement all its methods and export the creation fonction from the dll:
    IVehicleDynamics* createVehicleDynamics()
*/
class IVehicleDynamics
{
public:
    //! Destructor must be implemented in daughter classes
    /*! close record file, free memory, ... must be placed in this destructor

```

```
/*
virtual ~IVehicleDynamics() {}

/// Gives some information on a specific dynamic model
/*! This method is called before any other methods. It tells SCANeR what are the needs
     and capabilities of the dynamic model.
*/
virtual EVehicleDynFlags getDynFlags() = 0;

///Get the extension of the vehicle model file
virtual const char* getExtension() = 0;

/// Load simulation data (scenario, vehicle...)
virtual bool loadModel( const SimulationData& data ) = 0;

/// Return some vehicle information needed by the module to communicate properly with SCANeR
virtual bool getVehicleInfo(int vehicleIndex, VehicleInfo& info) = 0;

/// Initialisation or re-initialisation of the model
/*! During this function, the model should (re)initialize all data, for example by placing the vehicle
     at the exact position on the road, setting right speed, ...
     a statico dynamic initialization can be done.
*/
virtual bool initModel(const InitialData& initialData, double curTime) = 0;

/// Set the new input data
virtual bool setInput(const VehicleInput& input) = 0;

/// Set defects on the vehicle
virtual bool setDefects(int vehicleIndex, const Defects& defects) = 0;

/// During this function, the model should perform one step of simulation.
/*! deltaTime: time elapsed between 2 steps.
*/
virtual bool stepModel(double deltaTime) = 0;

/// Get the current output data
/*! When a vehicle is composed of several parts (like a truck and its trailer), the result is given per part.
*/
virtual bool getOutput(int vehicleIndex, VehicleOutput& output) = 0;

/// Ask model for custom data
/*! The model should fill a list of exported custom data (id/value).
     The modelhandler will send the data to SCANeR Channels 300+id.
     Return false if there is no data, true otherwise.
*/
#ifndef __RTX
    virtual bool getCustomData(std::vector<CustomData> &datas) { return false; }
#endif //__RTX

/// Some models may need to make some actions when the simulation is started/paused (freezing internal
timers,...)
/// Called when the simulation is paused
virtual bool pauseModel() = 0;

/// Called when the simulation is resumed
virtual bool resumeModel() = 0;

/// Return error message.
```

```
/*! This method is called each time 'false' was returned on one of the other methods.  
The string is then displayed in the SCANeR console as an Error message.  
errorCode: not used yet.  
*/  
virtual const char* getError( EModelErrorCode& errorCode ) = 0;  
  
/// Apply external force to the model  
/*! The external torser (applicationPoint, force, momentum) is defined in CoG frame  
*/  
virtual void applyForce(int vehicleIndex, const stk::Torser& externalForce) {};  
  
/// Set the speed of the vehicle  
virtual bool setSpeed(double speedObligatory) {return true;}  
virtual void onIdle() {}  
};  
  
} // namespace cats  
} // namespace model  
  
typedef cats::model::IVehicleDynamics* (*FCT_MakeDynPlugin)();  
typedef void (*FCT_DestroyDynPlugin)(cats::model::IVehicleDynamics*);
```

XI.B - BUILDING A SIMULINK VEHICLE SAMPLE

Open the Simulink vehicle model sample located in:

<STUDIO_PATH>\<STUDIO_VERS>\APIs\VehicleDynamics\samples\VehicleDynamicsSimulink\sample.mdl

click "tools->real time workshop->build model"

This will build the project and produce a DLL containing the vehicle model.

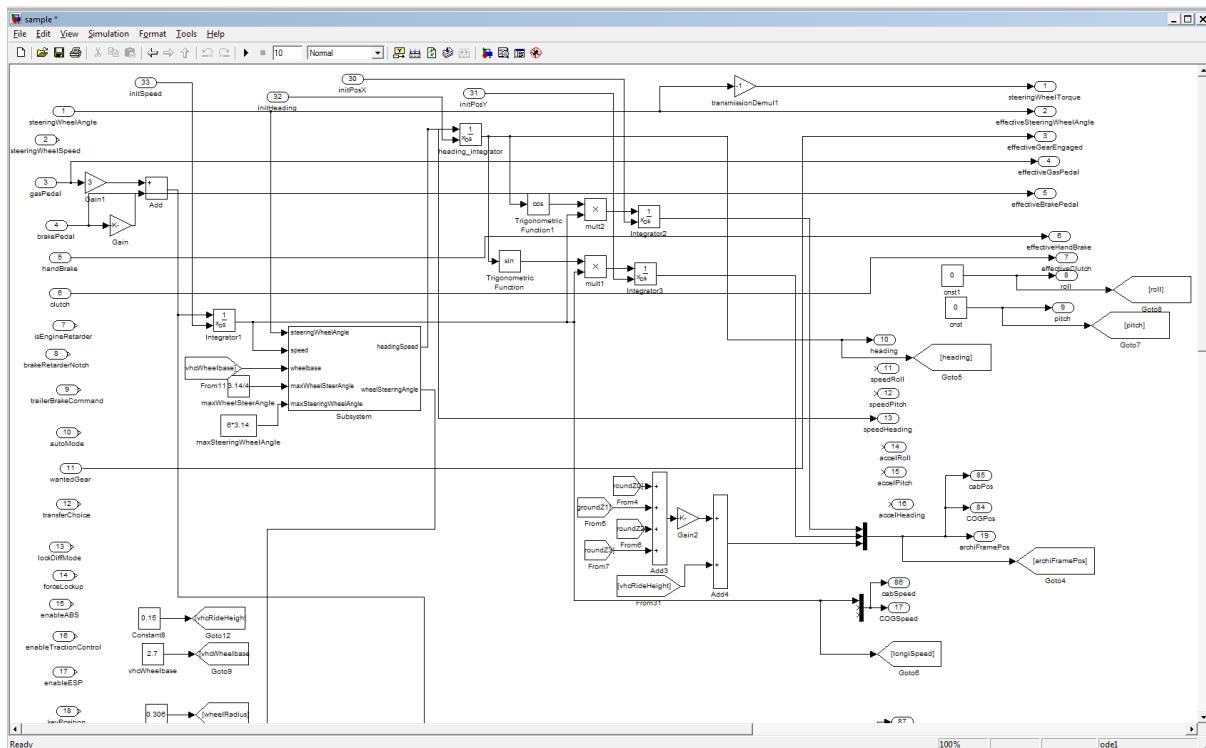


Illustration 89: Simulink

XI.B.1 - SIMULINK BUILD ENVIRONMENT

To compile a Simulink vehicle model for use with SCANeR™ studio, the C++ build environment must be correctly set up (see previous chapter).

Additionally, version of Matlab/Simulink, described in the "Environment" chapter (page 18), must be installed on the workstation, and compiler described in the same chapter must be registered as the compiler to use in the Simulink ERT.

XI.B.2 - USING THE EXTERNAL VEHICLE MODEL

Start SCANeR™ studio and go to the "vehicle" mode.

Click "file->new vehicle" and select "External" in the model type.

Click "OK" to validate the vehicle logical information.

In the “External compiled DLL name” field, write the full path of the vehicle model DLL.

Save the vehicle, it can now be used as any other vehicle resource in SCANeR™ studio.

To test the model, click on the “performance acceleration” element in the “predefined runs” dock of the vehicle mode, wait for the run to complete and analyse the result with AnalyzingTool.

XI.B.3 - INPUTS AND OUTPUTS

The inputs of the vehicle model are described in the SHM documentation

<STUDIO_PATH>\<STUDIO_VERS>\doc\Shm.html

The outputs of the vehicle model are listed in the table below:

Vehicle output name	type	Unit	Description
steeringWheelTorque	floating point	N.m	torque sent to the steering wheel force feedback
effectiveSteeringWheelAngle	floating point	rad	steering wheel angle after regulation (if any, otherwise echo command)
effectiveGearEngaged	floating point		gear effectively engaged by the transmission (can be different from gear command)
effectiveGasPedal	floating point	0-1	gas pedal after regulation (if any, otherwise echo command)
effectiveBrakePedal	floating point	N	brake pedal force after regulation (if any, otherwise echo command)
effectiveHandBrake	floating point	N	hand brake force after regulation (if any, otherwise echo command)
effectiveClutch	floating point	0-1	clutch after regulation (if any, otherwise echo command)
roll	floating point	rad	roll angle of the vehicle body, positive when leaning right
pitch	floating point	rad	pitch angle of the vehicle body, positive when leaning forward
heading	floating point	rad	heading of the vehicle, 0 to the east, positive counter clockwise
speedRoll	floating point	rad.s-1	roll speed
speedPitch	floating point	rad.s-1	pitch speed
speedHeading	floating point	rad.s-1	heading speed
accelRoll	floating point	rad.s-2	roll acceleration
accelPitch	floating point	rad.s-2	pitch acceleration
accelHeading	floating point	rad.s-2	heading acceleration

Vehicle output name	type	Unit	Description
COGSpeed	vector3	m.s-1	speed vector at the vehicle centre of gravity in the vehicle frame of reference (X forward, Y left, Z up)
COGAccel	vector3	m.s-2	acceleration vector at the vehicle centre of gravity in the vehicle frame of reference (X forward, Y left, Z up)
archiFramePos	vector3	m	position vehicle undercarriage at the middle of the first axel in absolute frame
effectiveLights	integer		
effectiveWipers	integer		
effectiveHorn	integer		
engineState	integer		true if the engine is running
consumption	floating point	m3/s	fuel consumption
pollution	floating point	kg/s	CO2 output
engineSpeed	floating point	rad/s	engine speed
COGPos	vector3	m	position of the vehicle centre of gravity in the absolute frame
cabPos	vector3	m	position of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabSpeed	vector3	m.s-1	speed of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabAccel	vector3	m.s-2	acceleration of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
wheelbase	floating point	m	the vehicle total wheelbase
rideHeight	floating point	m	the ride height of the vehicle (at the rear axle)
cabSpeedRoll	floating point	rad.s-1	roll speed of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabSpeedPitch	floating point	rad.s-1	pitch speed of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabSpeedHeading	floating point	rad.s-1	heading speed of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabAccelRoll	floating point	rad.s-2	roll acceleration of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity

Vehicle output name	type	Unit	Description
cabAccelPitch	floating point	rad.s-2	pitch acceleration of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabAccelHeading	floating point	rad.s-2	heading acceleration of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabRoll	floating point	rad	roll of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabPitch	floating point	rad	pitch of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
cabHeading	floating point	rad	heading of the vehicle driver cabin (if it has one), otherwise, same as vehicle centre of gravity
customOutputs[128]	floating point		128 custom output values, names and units must be specified in the vehicle logical information
absHubPosition	vector3	m	position of the hub of the wheel in the global reference
hubPositionRoll	floating point	rad	Absolute roll of the wheels hub
hubPositionPitch	floating point	rad	Absolute pitch of the wheels hub
hubPositionHeading	floating point	rad	Absolute heading of the wheels hub
wheelRotation	floating point	rad	total rotation of the wheel since start of the simulation
wheelRotationSpeed	floating point	rad.s-1	rotation speed of the wheel
brakeTemperature	floating point	C	brake temperature
absIsActive	integer		ABS or ESP is currently active on this wheel
isInContact	integer		the wheel is in contact with the ground
LSR	floating point	[-1 1]	longitudinal slip ratio of the tire
slipAngle	floating point	rad	side slip angle of the tire
contactPos	vector3	m	position in absolute frame of the tire contact point
contactNorm	vector3		normal of the ground at the tire contact point
grip	floating point		grip of the rolling surface at the wheel contact point

Table 28: Outputs of the vehicle model

XI.B.4 - INTERACTION WITH THE GROUND

The vehicle dynamics model can query the ground surface geometry and properties (such as grip) using the IRoadQuery interface.

The IRoadQuery interface is passed to the vehicle model in the loadModel() method. The rq member of the SimulationData is a pointer to this interface.

During initialisation, for each wheel (or other part of the vehicle that can interact with the ground surface), a wheel sensor must be initialised with the IRoadQuery::initWheel() method. This function returns the unique index of the sensor, typically, there is at least one sensor per wheel. The sensor indices must be stored by the vehicle model to be used by subsequent ground interaction function calls.

Once the wheel sensor is initialised, there are 2 main ways of using it to query the ground surface.

```
class IRoadQuery
{
public:
    //Initialize a wheel

    //When this method is called, the plugin initializes a new wheel and returns an unique
    //identifier of the wheel

    //param enableRugosity, enable rugosity for this sensor, rugosity must also be enabled
    //when loading the database for this option to have effect

    virtual int initWheel(bool enableRugosity);

    //Return the number of ground properties (physical material description) in the database

    virtual int getGroundPropertiesCount() const;

    //Return the ground properties (physical material description) associated to this index

    //When this method is called, the plugin must return a const reference of
    //GroundProperties.

    //param index of the ground

    virtual const GroundProperties& getGroundProperties(int index) const ;

    //Make simple vertical picking at the given coordinates

    //When this method is called, the RQ will fill the RoadVal structure for the given wheel
    //id and for x and y coordinate of the RoadVal struct. return true if picking ok

    //param wheelId unique identifier of the wheel

    //param pickingPos X,Y,Z position of sensor above the surface

    //param out Output structure including group intersection position and ground type
    //informations.

    virtual bool pick(int wheelId, const stk::Vector3& pickingPos, RoadVal& out);

    //Make complex picking for the wheel: cutout is the list of RoadVal intersecting wheel
    //plan and ground. return true if cutout ok

    //param wheelId unique identifier of the wheel

    //param Wheel Input data of the wheel

    //param out vector of RoadVal for each output structure for other fields.

    virtual bool cutout (int wheelId, const RoadWheelData& wheel, std::vector<RoadVal>& out);

};
```

XI.B.4.i - Picking the ground

The first, and simplest way of querying the ground surface using a wheel sensor is to call the IRoadQuery::pick() method. This method takes a point (as global 3D coordinates) and returns the position, normal and properties of the ground directly under the sensor (Z is the vertical axis of the global referential so the contact point will be of a lower Z value compared to the Z value of the sensor position).

The returned RoadVal structure contains the ground intersection position, the normal to the ground at this point as well as the ground index that can be used in the getGroundProperties() method to retrieve the ground's physical attributes at this point (ex: the grip of the surface).

XI.B.4.ii - Ground cutout

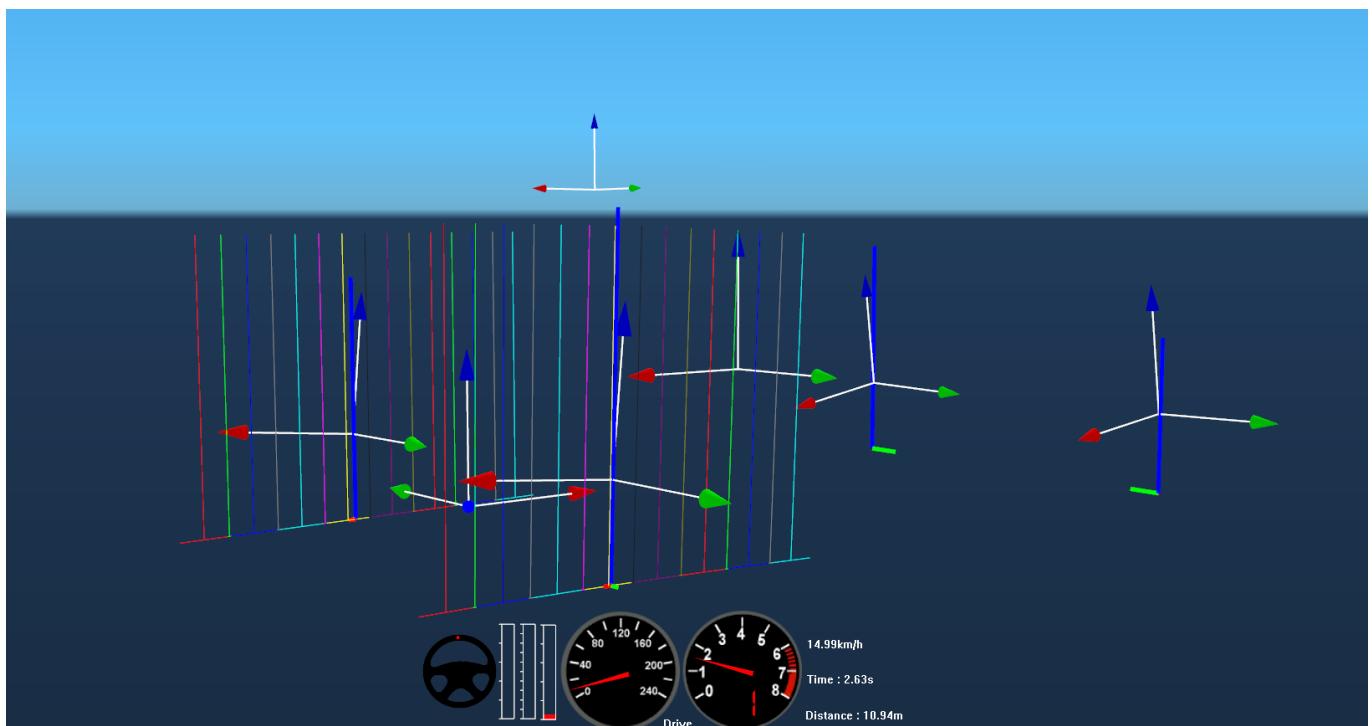


Illustration 90: Ground cutout of the two front wheels

For some tyre models, there can be more than one point of contact between the tyre and the ground, or, the point of contact on the ground surface can be offset by the tyre deformation. In this case, there needs to be more information than just one contact point to describe the ground at the location of the wheel.

One simple way to solve this problem is to instantiate several wheel sensors for each wheel and use the pick() method (as described earlier).

The other (recommended) way a tire can interact with the ground is by using the cutout() method.

A cutout of the road is the intersection of the ground's surface with a disk. The disk used for the cutout is specified in a RoadWheelData structure containing the position and orientation of the wheel's hub as well as the radius of the intersection disc.

The radius of the intersection disk should be greater than the maximum radius of the wheel's tire.

The position of the hub should have the correct heading (the heading is defined the same way as the vehicle heading (0 to the east, positive counter-clockwise seen from above, in radians). The roll and pitch of the hub are also defined the same way as the vehicle's.

The cutout between the wheel's disk and the ground surface is returned as a group of segments contained in a vector of RoadVal. The RoadVal intersection points are ordered from the back to the front of the wheel and the normals contained in the RoadVal point apply to the segment after the RoadVal point.

xii - VEHICLE DYNAMICS COMUDP API

xii.a - PRESENTATION

The Vehicle Dynamics ComUdp API is a SCANeR™ studio interface to integrate ComUdp vehicle models in SCANeR™ studio.



Any programming language can be used (*usage of UDP*).

The Vehicle Dynamics ComUdp is currently delivered as a Simulink block “ComVhcDyn” which is an S-function.

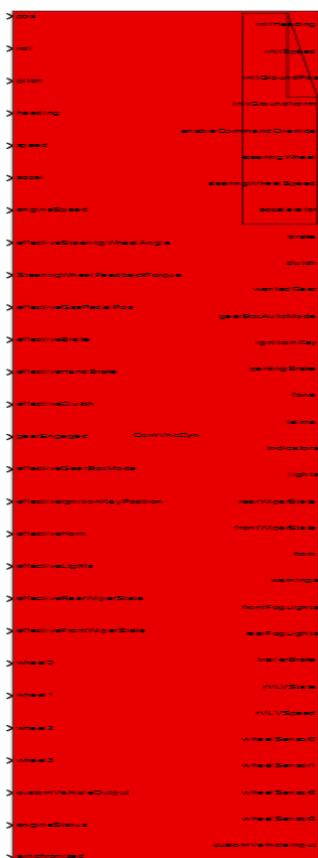


Illustration 91 :
ComVhcDyn block



The ComVhcDyn S-function uses Windows libraries dependencies for UDP communication. To set up this S-function dependencies double click on the block and select the tab “Libraries”.

XII.B - WORKFLOW

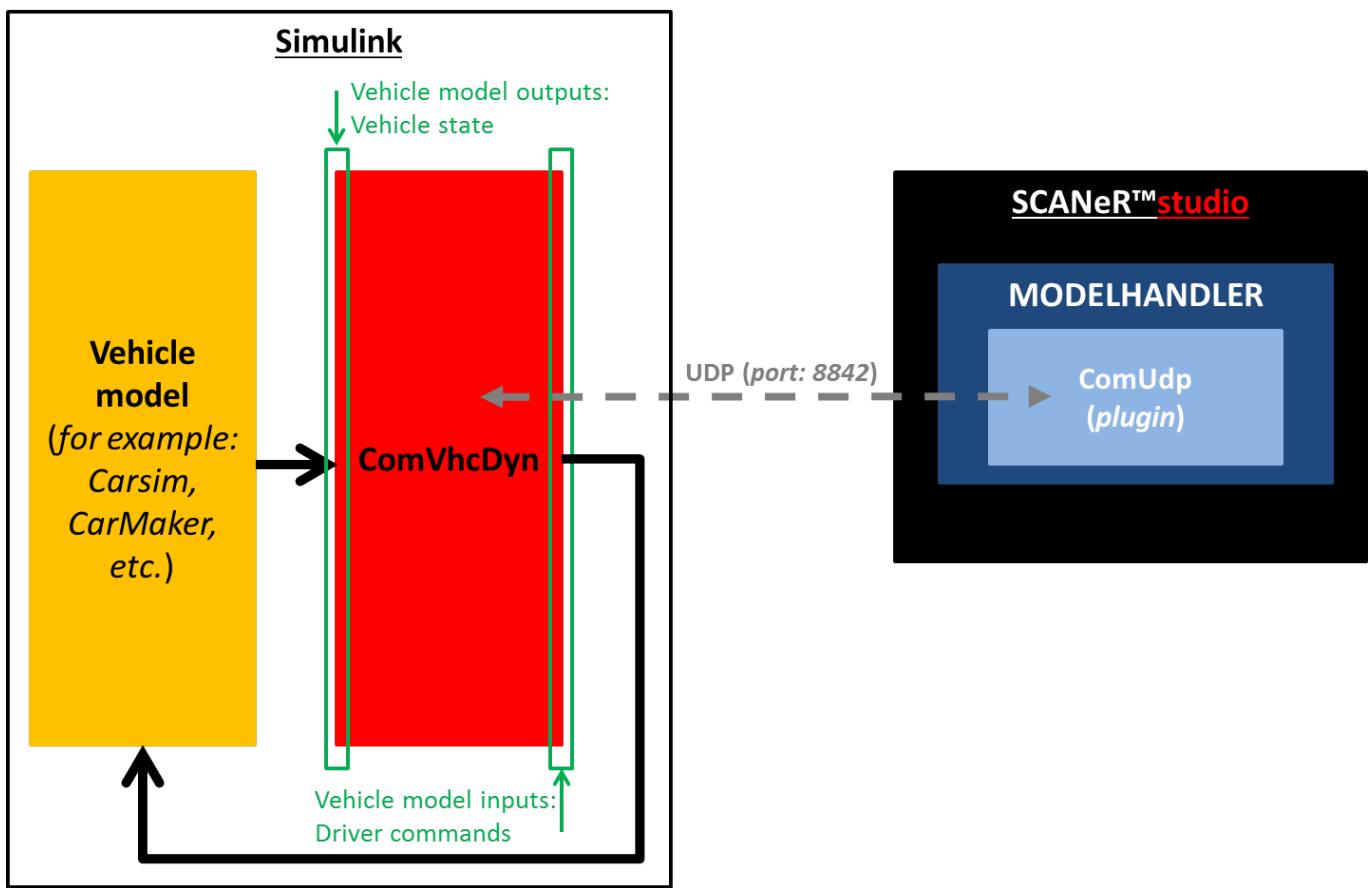


Illustration 92 : ComUdp workflow

The **orange block** is the vehicle model, it must be replaced by **your** vehicle block(s).

The **red block** is the ComVhcDyn S-function, it is used to communicate by UDP with the ComUdp vehicle model managed by the MODELHANDLER module (*as a plugin*).

It takes in inputs the vehicle model outputs (*which can be compared to the vehicle state*), the data are send to the MODELHANDLER module into SCANeR™ in order to be proceeded and then the outputs of the red block are the ouputs of the ComUdp vehicle model managed by the MODELHANDLER module (*which can be compared to the diver commands which can be used for driving assistance*).



The UDP communication port used by the ComVhcDyn S-function is **8842**

XII.C - REFERENCE FRAMES

All units used in SCANeR™ studio follow the IS international recommendations.

- The **ground reference**, is a Cartesian frame, centred at the 0 of the database, with **Z pointing upwards** from the ground. When viewed in the SCANeR™ studio supervision tool, the **X axis points to the East** and the **Y axis points to the North**. This frame can optionally be georeferenced (as described in the SCANeR™ studio Terrain editing tool). The ground reference position of the mouse is displayed in the SCANeR™ studio GUI in scenario mode at the top of the 3D supervision view.

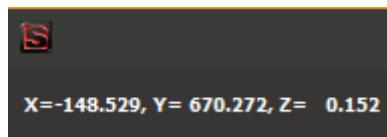


Illustration 93 : Ground reference

- To make interfacing of vehicle models with different vehicle references, the **measurement point** of the vehicle is specified in the ComUdp parameters GUI (*default values are 0, 0, 0*).

If the measurement point is $(0, 0, 0)$, the vehicle uses SCANeR's "vehicle coordinate system". The measurement point can be any point that is static relative to the vehicle body (*for example, the rear axle, front axle, geometric center or center of mass*).

The frame defined by the **measurement point** has the **X-axis facing forwards** and **Z-axis upwards and Y to the left**. The orientation of the frame is represented as **3 angular rotations**: **Heading** (*around the Z axis, positive to the left, 0 when facing along the X axis of the database, to the East*), **Pitch** (*around the Y axis, positive when the nose of the vehicle points downwards*) and **Roll** (*around the X axis, positive when the vehicle leans right*).

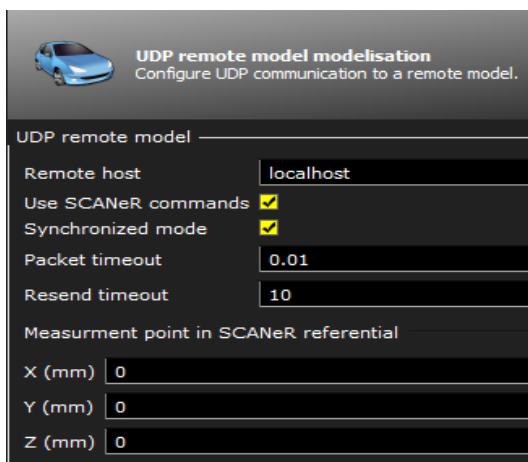


Illustration 94 : Measurement point

- The **initial position** and the **current position** of the vehicle are the X Y Z coordinates of the measurement point in the ground reference frame.

The vehicle's input includes the driver commands (*steering, gas pedal, brake pressure etc.*) as well as the ground properties for each wheel.

XII.D - VEHICLE MODEL INPUTS (DRIVER COMMANDS)

The outputs ComUDP are the driver commands (*the vehicle input*) as well as the initial position of the vehicle as specified in the scenario.

Model inputs	Type	Unit	Description
initHeading	double	rad	Initial vehicle heading
initSpeed	double	m/s	Initial vehicle speed
initGroundPos	double[3]	X Y Z	Initial position of measure point in the ground reference.
initGroundNorm	double[3]	X Y Z	Normal vector to the ground surface at the initial position. (0,0,1) for flat ground.
enableCommandOverride	boolean	0: FALSE 1: TRUE	Use driver commands from SCANeR. Can be used to switch from Human to Autonomous driver.
steeringWheel	double	rad	Steering wheel angle
steeringWheelSpeed	double	rad/s	Steering wheel angular speed
accelerator	double	[0,1]	Throttle pedal position
brake	double	N	Brake force
clutch	double	[0,1]	Clutch pedal position
wantedGear	int	0, 1, ..., 4, etc.	Wanted gear. <u>Note:</u> Used if gearBoxAutoMode is 0
gearBoxAutoMode	int	0: TautoModeManual, 1: TAUTOMODEREVERSE2, 2: TAUTOMODEREVERSE1, 3: TAUTOMODENEUTRAL, 4: TAUTOMODEPARK, 5: TAUTOMODELIMIT1, 6: TAUTOMODELIMIT2, 7: TAUTOMODELIMIT3, 8: TAUTOMODELIMIT4, 9: TAUTOMODERACE, 10: TAUTOMODEDRIVE, 11: TAUTOMODEOVERDRIVE, 12: Sequential.	Gear box mode.
ignitionKey	int	0: OFF, 1: ACCESSORIES, 2: RUN, 3: IGNITION.	Ignition key position
parkingBrake	double	N	Parking brake force
fowa	char	0: OFF 1: ON	Engine retarder (<i>for truck only</i>)
telma	double	0, 1, ... 4, or intermediate values	ElectroMagnetic brake position (<i>for truck only</i>)
indicators	int	-1: RIGHT,	Indicator lever position

		0: OFF, 1: LEFT	
lights	int	0: LIGHTS_OFF, 1: LIGHTS_PARKING, 2: LIGHTS_LOW, 3: LIGHTS_HIGH.	Lights lever position
rearWiperState	int	0: OFF, 1: INTERMITTENT_1, 2: INTERMITTENT_2, 3: INTERMITTENT_3, 4: LOW_SPEED, 5: HIGH_SPEED, 6: AUTOMATIC.	Rear wiper lever position
frontWiperState			Front wiper lever position
horn	char	0: OFF, 1: ON.	Horn state
warnings	char	0: OFF, 1: ON.	Warnings lights state
frontFogLights			Front fog lights state
rearFogLights			Rear fog lights state
trailerBrake	double	N	TrailerBrake state (<i>for truck only</i>)
rVLVState	int	Any int value	Speed Regulator/Limiter mode
rVLVSpeed	double	Any double value	Speed Regulator/Limiter setting
customVehicleInput	double[128]	Any double value]	Custom input values
WheelSensor0 wheelSensor1 wheelSensor2 wheelSensor3	contactPosition	double[3] X (m) Y (m) Z (m)	Wheel hub center position of contact point in the ground reference.
	contactNormal	double[3] X Y Z	Normal of ground at wheel's contact point in the ground reference. <u>Note:</u> 0, 0, 1 for flat ground.
	grip	double	Any double value (<i>1 is for dry asphalt</i>)
	groundIndex	int	Any int value (<i>refer to TERRAIN manual chapter “Ground edition”</i>)
	laneType	int	0=normal, 1=sidewalk, 2=ground, 3=obstacle, 4=parking, 5=island, 6=special, 7=out, 8=emergency stop.

XII.E - VEHICLE MODEL OUTPUTS (VEHICLE STATE)

The inputs to ComUDP are the vehicle state.

The vehicle state includes the position and engine speed.

The vehicle state also includes a copy of the driver controls **after they have been modified by the driving assistances** such as lane keeping, cruise control, automatic clutch, automatic gear selection or

other automation. In any case, the controls that are effectively used by the model should be output in the "effective controls" output values.

Model outputs	Type	Unit	Description
pos	double[3]	X (m) Y (m) Z (m)	Position of reference point of the vehicle in the measurement reference.
roll	double	rad	Orientation of the reference point of the vehicle in the measurement reference.
pitch	double	rad	
heading	double	rad	
speed	double[6]	X (m/s) Y (m/s) Z (m/s) Roll (rad/s) Pitch (rad/s) Heading (or Yaw) (rad/s)	Vehicle speeds
accel	double[6]	X (m/s ²) Y (m/s ²) Z (m/s ²) Roll (rad/s ²) Pitch (rad/s ²) Heading (or Yaw) (rad/s ²)	Vehicle accelerations
engineSpeed	double	rad/s	Engine speed
effecitveSteeringWheelAngle	double	rad	Effective steering wheel angle
SteeringWhellFeedbackTorque	double	N	Steering wheel torque
effectiveGasPedalPosition	double	[0, 1]	Effective throttle pedal
effectiveBrake	double	N	Effective brake pedal force
effectiveHandBrake	double	N	Effective parking brake force
effectiveClutch	double	[0, 1]	Effective clutch pedal
gearEngaged	int	[0 to N]	Current gear number
effectiveGearBoxMode	int	0: TautoModeManual, 1: TAUTOmodeReverse2, 2: TAUTOmodeReverse1, 3: TautoModeNeutral, 4: TautoModePark, 5: TAUTOmodeLimit1, 6: TAUTOmodeLimit2, 7: TAUTOmodeLimit3, 8: TAUTOmodeLimit4, 9: TautoModeRace, 10: TautoModeDrive, 11: TautoModeOverDrive, 12: Sequential.	Current gearbox mode
effectivelgnitionKeyPosition	int	0: OFF 1: ACCESSORIES	Effective ignition key

		2: RUN 3: IGNITION	
effectiveHorn	char	0: OFF 1: ON	Horn state
effectiveLights	int	bitset	Lights status of the vehicle. For bitset details refer to the Network.html.
effectiveRearWiperState	int	0: WIPERS_OFF, 1: WIPERS_INTERMITTENT_1,	The state of the rear wiper of the vehicle.
effectiveFrontWiperState	int	2: WIPERS_INTERMITTENT_2, 3: WIPERS_INTERMITTENT_3, 4: WIPERS_LOW_SPEED, 5: WIPERS_HIGH_SPEED, 6: WIPERS_AUTOMATIC.	The state of the front wiper of the vehicle
wheel0 wheel1 wheel2 wheel3	double[6]	X (m)	Position of wheel hub
		Y (m)	
		Z (m)	
		Roll (rad)	Camber
		Pitch (rad)	When suspension pivots at point behind wheel
		Heading (or Yaw) (rad)	Steering
		double	Wheel rotation speed
wheel0	double	rad/s	Wheel rotation speed
wheel1	double	rad	Current wheel rotation angle
wheel2	double		LSR (<i>Longitudinal Slip Ratio</i>) of the wheel.
wheel3	double	rad	Wheel Slip Angle (<i>angle in radians between actual speed vector of the wheel and wheel orientation</i>).
customVehicleOutput	double[128] [0 to N]		Range of data reserved for the user.
engineStatus	char	0: OFF 1: ON	Engine running flag
synchronized	char	0: Non-synchronized 1: Synchronized	In synchronized mode, the model will indefinitely wait for the 1st packet from MODELHANDLER.

XII.F - SAMPLE “REMOTE VEHICLE MODEL”

XII.F.1 - PRESENTATION

The RemoteVehicleModel Simulink sample is available under:

<STUDIO_PATH>\<STUDIO_VERS>\APIs\samples\VehicleDynamics\RemoteVehicleModel\RemoteVehicleModel.mdl



This model can be run directly inside Simulink or can be compiled to a Windows program (.exe). Its default compilation target is “ert.tlc” but currently the ComVhcDyn S-function uses Windows dependencies so it can only be executed on a Windows Platform.

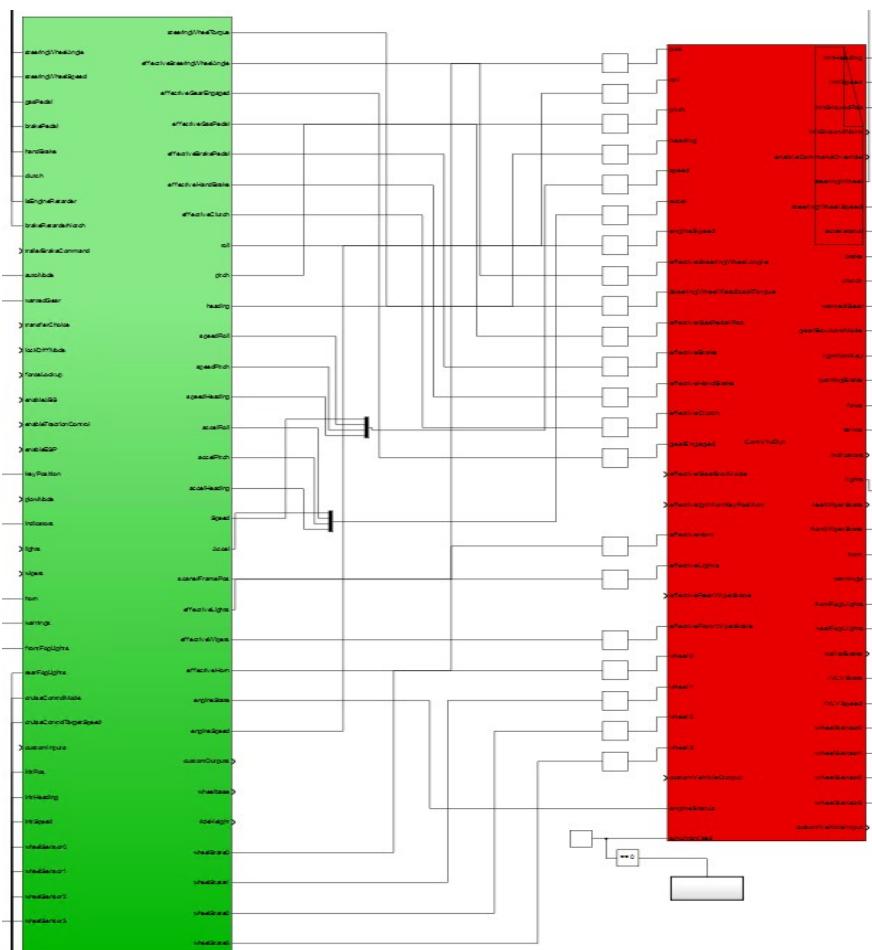


Illustration 95 : RemoteVehicleModel sample

XII.F.2 - BUILD AND RUN THE SAMPLE

3 steps are needed to use the RemoteVehicleModel sample:

1. Setup SCANeR™ studio,
2. Setup the RemoteVehicleModel Simulink model,
3. Run a Simulation.

XII.F.2.i - Setup SCANeR™ studio

A ComUdp vehicle model is delivered with the installer of SCANeR™ studio, it is named RemoteVehicleModel, it is available into the vehicle resources under category ComUdp.

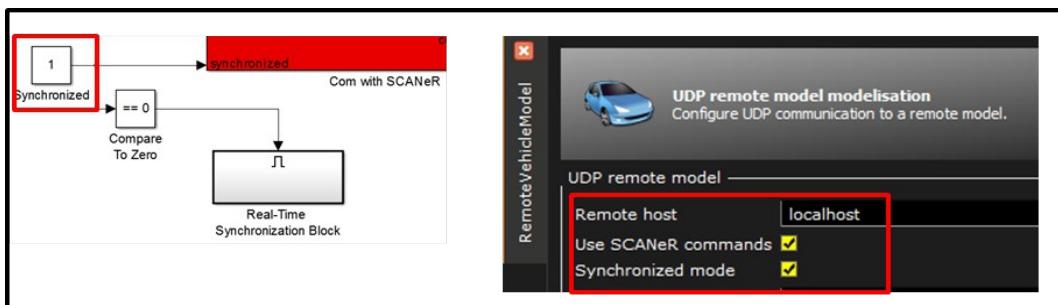
1. To test the Vehicle Dynamics ComUdp API open the vehicle RemoteVehicleModel under the VEHICLE mode of SCANeR™ studio and create a copy (*save it as*).

Then, on the copy, you need at least to fill the following parameters:

A - “Remote host”, is the IP address of the computer where the Simulink model is running (*if Simulink and SCANeR™ studio run on the same computer leave the default value “localhost”*).

Note: The UDP communication port is 8842.

B - “Synchronized mode”, if checked then the “**synchronized**” input port of the “**Com with SCANeR**” block must be equal to 1, 0 otherwise.



OR

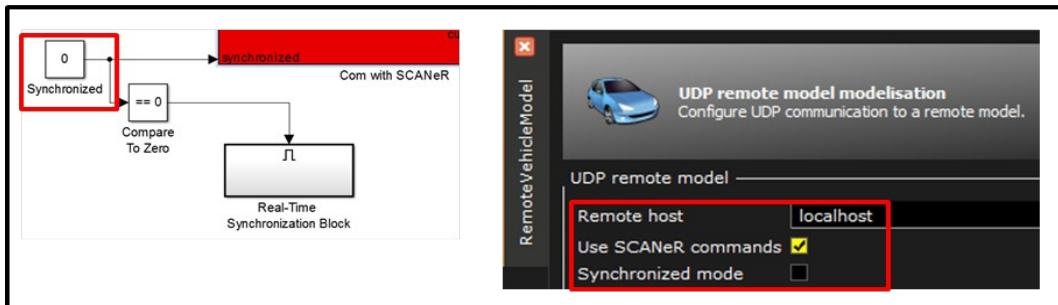


Illustration 96 : Remote host and Synchronized mode settings

Note: When running the model in **synchronized mode**, the model must be started before SCANeR™ simulation. The model will indefinitely wait for the first data packet from **MODELHANDLER**. If one of the communication endpoint stops sending data, the other one will stop after a specified timeout (*10 seconds for the model, UDPPresentTimeout seconds for SCANeR™ studio*). If the model stops, Simulink will display an error message and stop its simulation. If **MODELHANDLER** stops, the SCANeR™ simulation stops.

2. Under the SCENARIO mode, create a scenario that uses the ComUdp vehicle model you created in 1.

XII.F.2.ii - Setup the RemoteVehicleModel Simulink model

1. Setup Matlab/Simulink environment to build S-function:

Select a compatible compiler for your Matlab version using the command “**mex –setup**”

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>> mex -setup

Welcome to mex -setup. This utility will help you set up
a default compiler. For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2011a/win32.html

Please choose your compiler for building MEX-files:

fx Would you like mex to locate installed compilers [y]/n? y
```

Illustration 97 : Select a compiler using mex -setup command

2. Build the ComVhcDyn S-function:

Double click on the “ComVhcDyn” block and click on the “Build” button:

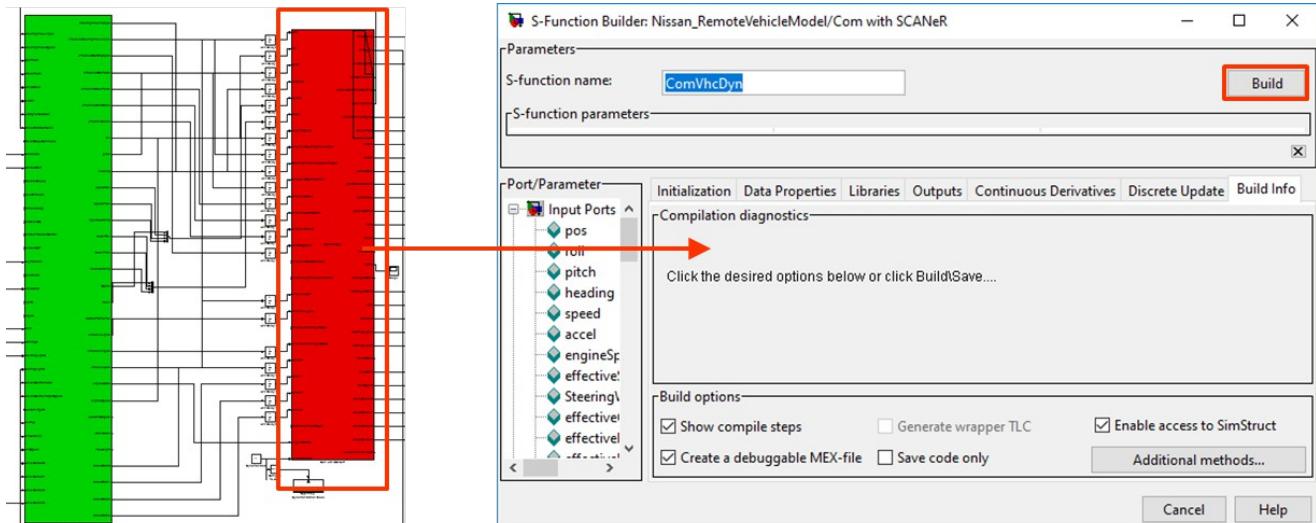


Illustration 98 : Build the ComVhcDyn S-function

3. Build the sfun_rtttime S-function:

```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
fx >> mex sfun_rtttime.c
```

Illustration 99 : Build the sfun_rtttime S-function

Into the Matlab Command Window execute the following command: **mex sfunc_rtttime.c**

XII.F.2.iii - Run a Simulation

- **Synchronized mode**, then the Simulink model must be started before SCANeR™ simulation. The model will indefinitely wait for the first data packet from **MODELHANDLER**. If one of the communication endpoint stops sending data, the other one will stop after a specified timeout (*10 seconds for the model, UDPresentTimeout seconds for SCANeR™ studio*). If the model stops, Simulink will display an error message and stop its simulation. If **MODELHANDLER** stops, the SCANeR™ simulation stops.
- **Non-Synchronized mode**, there is no specific order, the last received/emitted UDP packet is used.

XII.G - RUN MY VEHICLE MODEL IN REAL-TIME OR IN NON REAL-TIME

4 solutions can currently be used with SCANeR:

- The Vehicle model and **MODELHANDLER** run at their own frequencies,
- The simulation time is scheduled by **MODELHANDLER**,
- The simulation time is scheduled by the Vehicle model,
- The simulation time is scheduled by the **OFFLINESCHEDULER**.

XII.G.1 - REAL-TIME IN NON SYNCHRONIZED MODE

The Vehicle model and **MODELHANDLER** module run at their own frequencies.

The last received/emitted UDP packets are used.

The “**synchronized**” field of the ComVhcDyn S-function must be set to **0** and the option “**Synchronized mode**” in the ComUdp vehicle model must be **unchecked**.

The **MODELHANDLER** option “**Enable external scheduling**” must be unchecked.

XII.G.2 - REAL-TIME IN SYNCHRONIZED MODE

The **MODELHANDLER** module schedules the simulation.

The Vehicle model period must be the same as the MODELHANDLER module.

The “**synchronized**” field of the ComVhcDyn S-function must be set to **1** and the option “**Synchronized mode**” in the ComUdp vehicle model must be **checked**.

The **MODELHANDLER** option “**Enable external scheduling**” must be unchecked.

XII.G.3 - REAL-TIME IN SYNCHRONIZED MODE WITH AN EXTERNAL SCHEDULING

The Vehicle model schedules the simulation.

The “**synchronized**” field of the ComVhcDyn S-function must be set to **1** and the option “**Synchronized mode**” in the ComUdp vehicle model must be **checked**.

The **MODELHANDLER** option “**Enable external scheduling**” must be checked.

XII.G.4 - Non Real-time in Synchronized Mode

The **OFFLINESCHEDULER** module schedules the simulation.

The “**synchronized**” field of the ComVhcDyn S-function must be set to **1** and the option “**Synchronized mode**” in the ComUdp vehicle model must be **checked**.

The **MODELHANDLER** option “**Enable external scheduling**” must be unchecked.

The frequency (*period*) specified into the **OFFLINESCHEDULER** module for the **MODELHANDLER** module must be the same as the period defined into the Vehicle model.

XII.H - How to run an ADAS into my vehicle model?

There are 2 solutions:

1. Use the SCANeR API.

OR

2. Use the RTGateway module (under license, if you are interested in its evaluation please send an e-mail to support-scaner@oktal.fr).

XII.H.1 - Use the SCANeR API

XII.H.1.i - Workflow

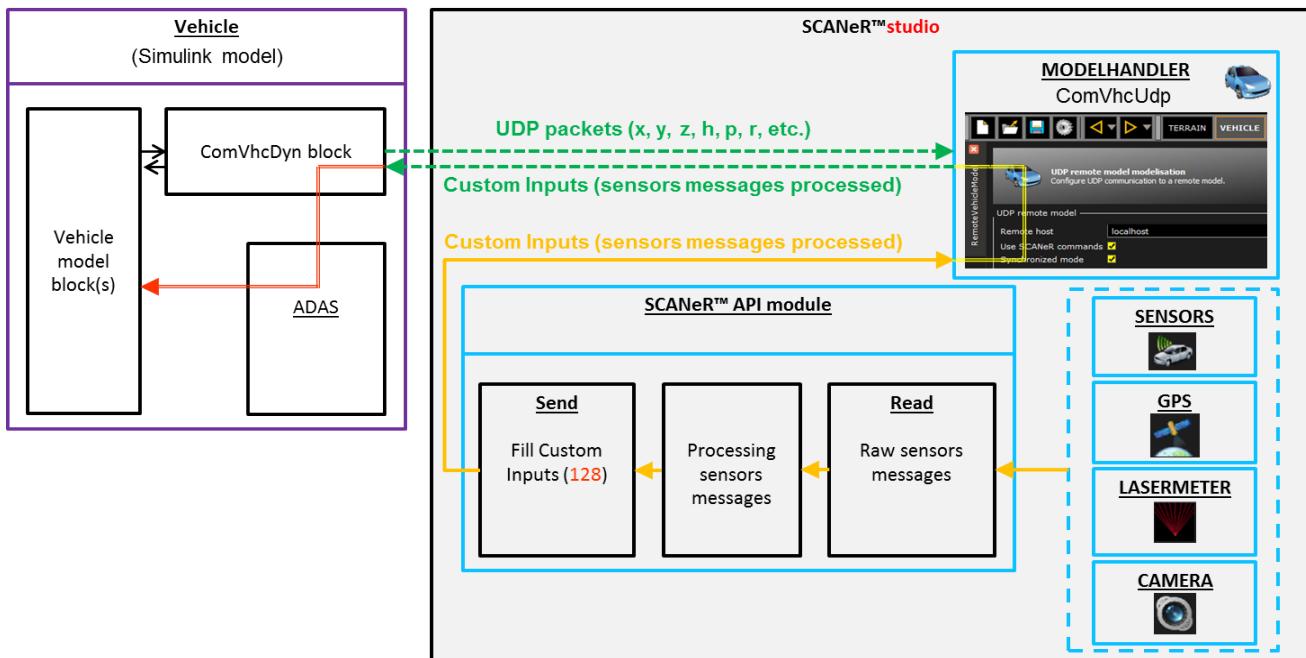


Illustration 100 : ADAS with SCANeR API

Into SCANeR™ studio create a scenario using a ComUdp vehicle model. This vehicle communicates with ComVhcDyn S-function.

For ADAS, the ComUdp vehicle model must be equipped with sensors.

The **SCANeR API** Module reads the sensors raw data and send these on the Custom Inputs of the vehicle model ComUdp.

The 'customInput' field which will contain the sensors data filled by the **SCANeR API** Module is available into the "Driver commands" of the ComVhcDyn S-function.

Limitation: Only 128 fields available.

XII.H.2 - USE THE RTGATEWAY MODULE

XII.H.2.i - Workflow

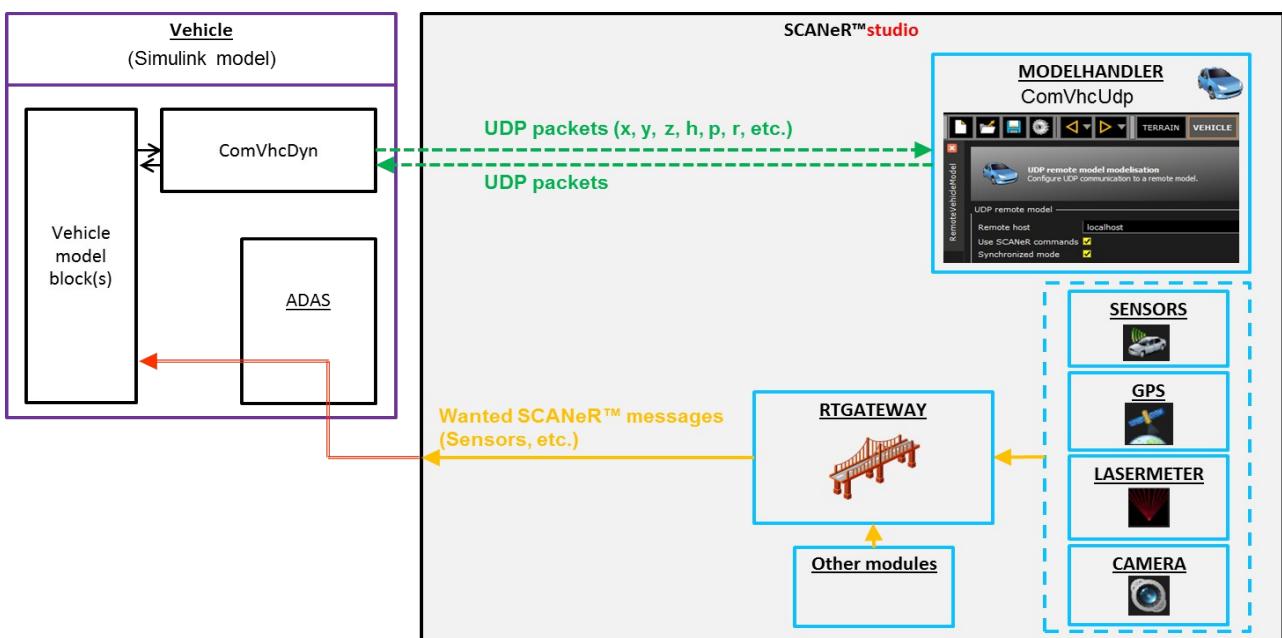


Illustration 101 : ADAS with RTGateway

Into SCANEr™ studio create a scenario using a ComUdp vehicle model and add to your SCANEr configuration the **RTGATEWAY** module.

For ADAS, the ComUdp vehicle model must be equipped with sensors.

The module **RTGATEWAY** sends the sensors raw data though UDP.

The Simulink model equipped with the ComVhcDyn S-function reads the UDP sensors raw data sent by **RTGATEWAY**. Based on the sensors raw data the ADAS can be developed into the same Simulink model.

XIII - MOTION API

XIII.A - API PRESENTATION

SCANeR™ studio supports the use of a motion platform in a simulator. A motion platform is a mechanical system capable of moving the driver's cabin to simulate the sensation of motion.

MOTION is the SCANeR™ studio module that interfaces the vehicle dynamics model with the motion platform. The use of this module is described in the MOTION chapter of the SCANeRstudio_SIMULATION_UserManual documentation.

The **Motion Strategy** is the software component that computes the motion platform command based on the movement of the virtual vehicle.

The **Motion Communication** is the software component that implements the communication protocol between the SCANeR MOTION module and the motion platform.

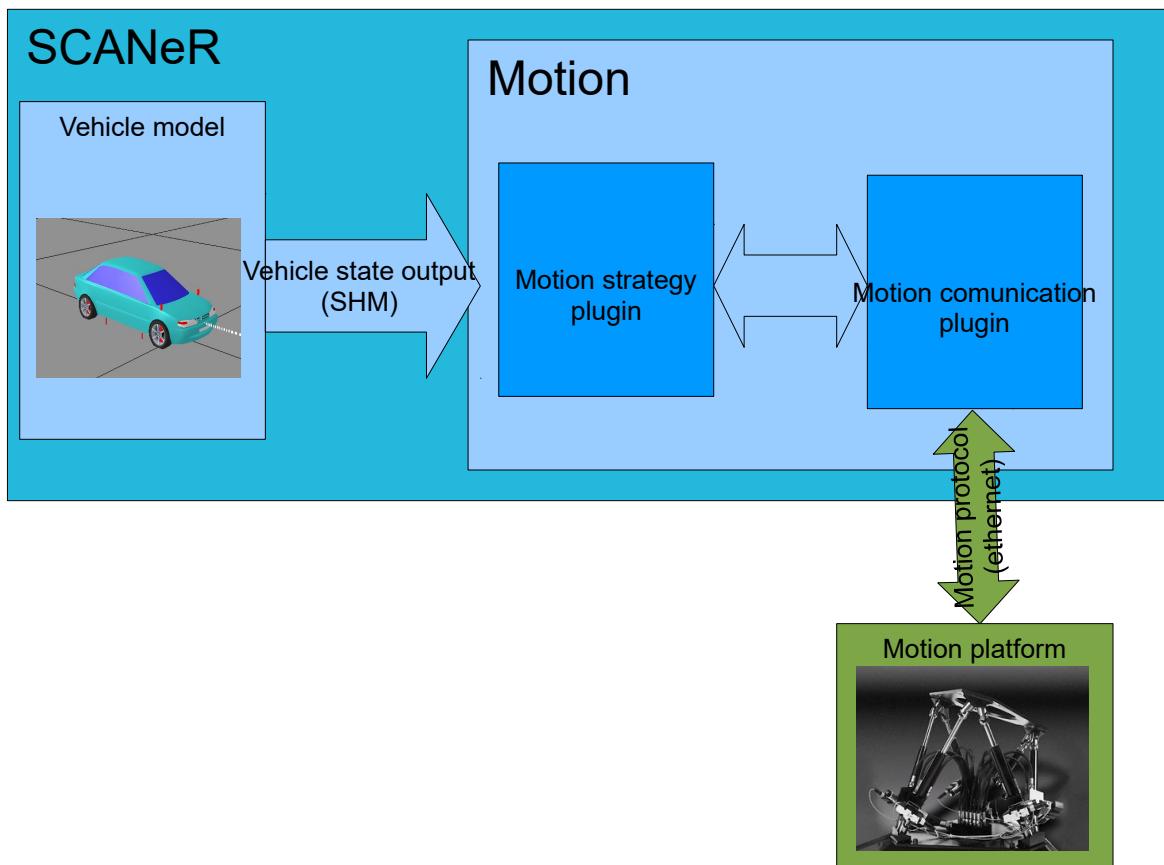


Illustration 102: Overview of motion platform strategy

This document explains how to use the SCANeR™ studio Motion API to develop a custom Motion Strategy.

XIII.B - DESCRIPTION

XIII.B.1 - INTERFACE

This paragraph describes the interface of the API for motion cueing strategies which is common to the C and Simulink implementations of the API.

XIII.B.1.i - Inputs

The structure of « MotionInput » is in
“%STUDIO_PATH\SCANeRstudio_1.6\APIs\include\MotionControl\MotionInput.h”

Input	Type	Description
speed	Stk::Mark	Speed of the measured point in the absolute frame.
acceleration	Stk::Mark	Acceleration of the mesures point in the absolute frame.
position	Stk::Mark	Potion of the vehicle CDG in the absolute frame.
groundPitch	double	Angle in radians of the ground slope, positive downhill.
groundRoll	double	Angle in radians of the ground banking, positive to the left.
GroundProperty.ground Grip	double[4]	The grip of the surface under the wheel
GroundProperty.ground Granularity	double[4]	Granularity of the ground surface as a proportion (0-1) for each wheel.
GroundProperty.ground Type	int[4]	Type of the ground under the wheel
GroundProperty.ground Index	int[4]	Index in the array of ground properties og the current RND file.
GroundProperty.laneType	int[4]	Type of the lane under the wheel
Frequency	{frequency, amplitude}	These values describe a sine wave vibration to be applied to the platform as a special effect. For each sine wave, the frequency and amplitude is specified. There can be a different amplitude for each axis. This type of special effect is supported natively by certain Bosch platforms.
specialEffects	Double[5]	These are the commands to the native special effects as supported by certain Bosch platforms.
effectivePos	Stk::Mark	The effective position of the motion platform
effectiveSpeed	Stk::Mark	The effective speed of the motion platform

Input	Type	Description
effectiveAcceleration	Stk::Mark	The effective acceleration of the motion platform
effectiveXYPos	Stk::Vector2	The effective position of the XY rails
effectiveXYSpeed	Stk::Vector2	The effective speed of the XY rails
effectiveXYPos	Stk::Vector2	The effective acceleration of the XY rails
vehiclePlayerCommand	Play/Pause/Seek	This value is only used when the vehicle is a "VehicleDynamicsPlayer". The value is the command the VehiclePlayer GUI is sending to the VehicleDynamicsPlayer.
vehiclePlayerState	Play/Pause/Seek	This value is only used when the vehicle is a "VehicleDynamicsPlayer". This is the effective state of the VehicleDynamicsPlayer. This state can be different from the command if the vehicle player is not authorized to do the transition to the new state.
platformOffset	Stk::Mark	The offset command for the motion platform. Usually, this offset is sent by the scenario script to request the platform prepositions itself in a position favourable to the current manoeuvre.
linearActuatorsOffset	Stk::Vector3	The offset command for the X/Y rail for prepositionning
Roll/pitch/heading ActuatorsOffset	double	Preposition for the yaw table if there is one
vehiclePlayerCommand	enumerate	When a GUI that commands play/pause/stop to the simulation, this is the current command (this parameter can be ignored)
vehiclePlayerState	enumerate	When a GUI that commands play/pause/stop to the simulation, this is the current state (this parameter can be ignored)
custom	double[128]	The values of these inputs are the current values of the vehicle variables that were chosen in the Motion configuration. See the Motion configuration chapter for instructions to choose the custom input list.

Table 29: motion strategy inputs

XIII.B.1.ii - Outputs

The structure <MotionStrategyOutput> is in: "%STUDIO_PATH%\SCANeRstudio_1.6\APIs\include\MotionPlugin\MotionStrategyStruct.h"

Output	Type	Description
modulatedVehicleAcceleration	stk::Mark	Acceleration of the vehicle sent to the motion platform. Fill this with the acceleration of the vehicle only if you want to use the platform cueing algorithm. This should be 0 if you are doing your own cueing.

Output	Type	Description
modulatedVehicleSpeed	stk::Mark	Speed of the vehicle sent to the motion platform. Fill this with the acceleration of the vehicle only if you want to use the platform cueing algorithm. This should be 0 if you are doing your own cueing.
modulatedVehiclePosition	stk::Mark	Position of the vehicle sent to the motion platform. Fill this with the acceleration of the vehicle only if you want to use the platform cueing algorithm. This should be 0 if you are doing your own cueing.
directPosition	stk::Mark	Command position of the motion platform
directLinearPosition	stk::Vector3	Position command of the XY linear actuators
directLinearSpeed	stk::Vector3	Not used
directLinearAcceleration	stk::Vector3	Not used
directRollActuatorPosition	double	Reserved for future use, leave 0
directPitchActuatorPosition	double	Reserved for future use, leave 0
directHeadingActuatorPosition	double	The command to the yaw actuator (for example the yaw table if there is one)
customData	double[128]	Custom values. These values are not used, they are just logged in the results file for analysis. It is possible to name these values in the Motion configuration dialog.
specialEffect	0 or 1	Enable special effect
bump	double[5]	Only valid for Rexroth motion platforms. Values passed on the special effects filters. See Rexroth documentation.
frequency	double[20]	Only valid for Rexroth motion platforms. Values passed on the special effects filters. See Rexroth documentation.
gain	stk::Vector3[20]	Only valid for Rexroth motion platforms. Values passed on the special effects filters. See Rexroth documentation.
authorisePlay	bool	This value is only used when the vehicle is a VehicleDynamicsPlayer. When this boolean is true, the VehicleDynamicsPlayer is authorised to transition from pause or seek state to the play state.

Output	Type	Description
authorizePause	bool	This value is only used when the vehicle is a VehicleDynamicsPlayer. When this boolean is true, the VehicleDynamicsPlayer is authorised to transition from play or seek state to the pause state.
authorizeSeek	bool	This value is only used when the vehicle is a VehicleDynamicsPlayer. When this boolean is true, the VehicleDynamicsPlayer is authorised to transition from play or pause state to the seek state.

XIII.B.1.iii - Table 30: motion strategy outputs

XIII.B.2 - TYPES OF STRATEGIES

There are two ways to use the SCANeR Motion Strategy API. One way is to do a “Modulation Strategy”, and the other is to do a “Position control strategy”. These two methods are completely different and you should carefully consider which one to use.

XIII.B.2.i.a - Modulation strategies

Modulation strategies simply take the position/speed/acceleration of the virtual vehicle model as input, modulate them, add special effects, and pass them on to the Platform cueing algorithm. The “default” strategy that comes as standard with SCANeR™ studio is a modulation strategy. This kind of strategy can be implemented to keep the motion platform default cueing algorithm overall functioning but add special effects specific to the project.

This type of strategy only uses the motion platform direct input only to add vibrations and other effects. Most of the movement will be done by the motion platform cueing algorithm based on the position/speed/acceleration of the virtual vehicle.

XIII.B.2.i.b - Position control strategies

Position control strategies completely bypass the platform standard motion cueing algorithms. These strategies do NOT forward the position/speed/acceleration of the virtual vehicle model, they compute the motion platform command position directly from the virtual vehicle state and send the command to the platform.

XIII.B.3 - THE C API

XIII.B.3.i - Principle

To create a custom strategy it is necessary to create a new class that inherits from *MotionStrategyInterface* and to re-implement virtual functions that are described in the next chapter.

Example:

```
class MotionSampleStrategy : public MotionStrategyInterface
```

XIII.B.3.ii - Compilation environment

The C motion strategy samples, like all other SCANeR samples, is meant to be compiled with compiler described in the "Environment" chapter (page 18). If an other compiler is used (from an other vendor or a different version of the compiler), use a statically linked C runtime.

The strategy should be compiled with the same version of the SDK as the version of SCANeR™ studio that is going to be used at runtime.

XIII.B.3.iii - Functions

The SCANeR Motion API contains the following functions:

XIII.B.3.iii.a - Constructor

```
MotionSampleStrategy(  
    const char* configFile,  
    const PlatformInfo & pfInfo,  
    IMotionRoadQuery* rq  
) ;
```

The constructor receives the path to the configuration file of the strategy. This path is the one specified in the Motion options dialogue. This file can contain any options you might need. SCANeR does not impose a format for this configuration file, it is up to the strategy developer to determine its format and how to read it in the constructor.

configFile: the complete path to the configuration file of the strategy. This path is the one specified in the Motion options dialogue.

pfInfo: A structure describing the motion platform is also passed to the constructor. This contains information about the platform dimensions and can be used to modulate the strategy outputs based on the platform limitations (ex: the length of the).

rq: an interface to query the logical road content. This can be used to compute platform prepositioning.

XIII.B.3.iii.b - Compute

```
virtual bool compute(  
    double deltaT,  
    const MotionStrategyInput &modelToPlatform,  
    const MotionComOutput &motionPos,  
    MotionStrategyOutput &out  
) ;
```

This is the function where all the cueing filters should be done. The inputs and outputs of this function are documented in the “inputs/outputs” list. The outputs should be computed based on the input values and any integration done in the strategy should be based on the deltaT time sample length.

DeltaT: the time since the last call to compute. This is usually equal to the inverse of the motion module running frequency, as specified by the “-f” command line option of motion.exe. If the strategy contains values to be integrated or derived, this time step should be used.

ModelToPlatform: the inputs of the strategy for this time step. See Inputs definition for the meaning of each input value.

MotionPos: the current state of the platform.

Out: the outputs of the motion strategy. The compute() function should fill this structure. See Outputs reference for the meaning of each of these outputs.

XIII.B.3.iii.c - Destructor

```
virtual ~MotionSampleStrategy();
```

All resources of the strategy should be released here.

XIII.B.4 - THE SIMULINK API

XIII.B.4.i - Using the template make file

To create a new SCANeR motion strategy, copy one of the samples and rename it.

To compile a SCANeR motion strategy, you must first make sure that the <STUDIO_PATH>\<STUDIO_VERS>\APIs\Motion\MotionSimulinkStrategy\MotionStrategy.tmf file is selected as the “template make file” in the model parameters.

Tip: some times, for this option to be correctly set, you need to choose a different TMF and then select “MotionStrategy.tmf” again.

XIII.B.4.ii - Simulink version

The mdl samples are edited with Simulink version described in the "Environment" chapter (page 18) and should compile “as is”. If a different version is used, it is possible the model needs to be modified to work correctly and/or compile.

XIII.B.4.iii - C runtime version

As for strategies developed with the C API, strategies compiled with Simulink are meant to be compiled by compiler described in the "Environment" chapter (page 18). This compiler must be registered as the target compiler of Simulink (see Simulink documentation for how to do this). If a different compiler is used, or a different version, enable linking with the static version of the C runtime

For example, with a different version of Visual Studio, to enable linking with the static C runtime libraries, uncomment the following lines in the *.tmf file:

```
#CFLAGS = $(CFLAGS:-MD=-MT)  
#CPPFLAGS = $(CPPFLAGS:-MD=-MT)  
#CFLAGS = $(CFLAGS:-D_DLL=)  
#CPPFLAGS = $(CFLAGS:-D_DLL=)
```

XIII.B.4.iv - valid motion strategy model

To compile a valid strategy model, all the following conditions must be met:

- MotionStrategy.tmf template make file must be set.
- All the inputs and outputs present in the samples must be present, with the same name and port width.

Once these conditions are verified, compile the model (Ctrl+B).

XIII.B.5 - SAMPLES

The motion API contains samples in C and in Simulink that can be used to base your Strategy developments on.

XIII.B.5.i - MotionSampleStrategy

location: <STUDIO_PATH>\<STUDIO_VERS>\APIs\Motion\MotionSampleStrategy

This sample is a simple “pass through”. It is the simplest form of “modulation strategy”. It forwards the position/speed/acceleration of the virtual vehicle model to the motion platform cueing algorithm. It is similar to the default strategy that is used by default by motion but supports none of the parameters (low speed filter, initial movement ramp...). To add features to this strategy, you must copy the project and modify the C code.

XIII.B.5.ii - PassThrough

location: <STUDIO_PATH>\<STUDIO_VERS>\APIs\Motion\MotionSimulinkStrategy\PassThrough.mdl

this is a Simulink version of the PassThrough C Strategy.

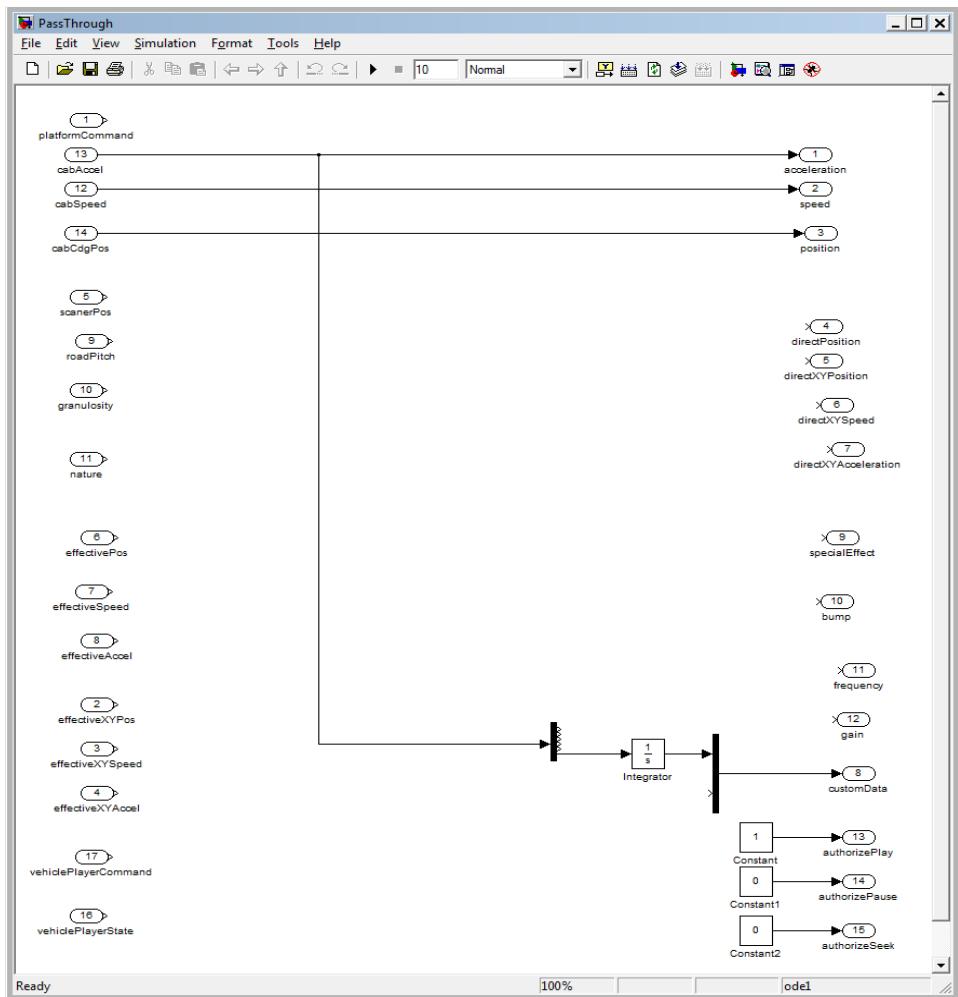


Illustration 103: PassThrough Simulink Motion Strategy sample

XIII.B.5.iii - DirectReply

location: <STUDIO_PATH>\<STUDIO_VERS>\APIs\Motion\MotionSimulinkStrategy\DirectReply.mdl

Contrary to the PassThrough strategy, this is a “position control strategy”. This strategy is peculiar because it assumes that the motion platform position is computed by the vehicle model and passes on the motion platform positions straight from the vehicle model to the platform.

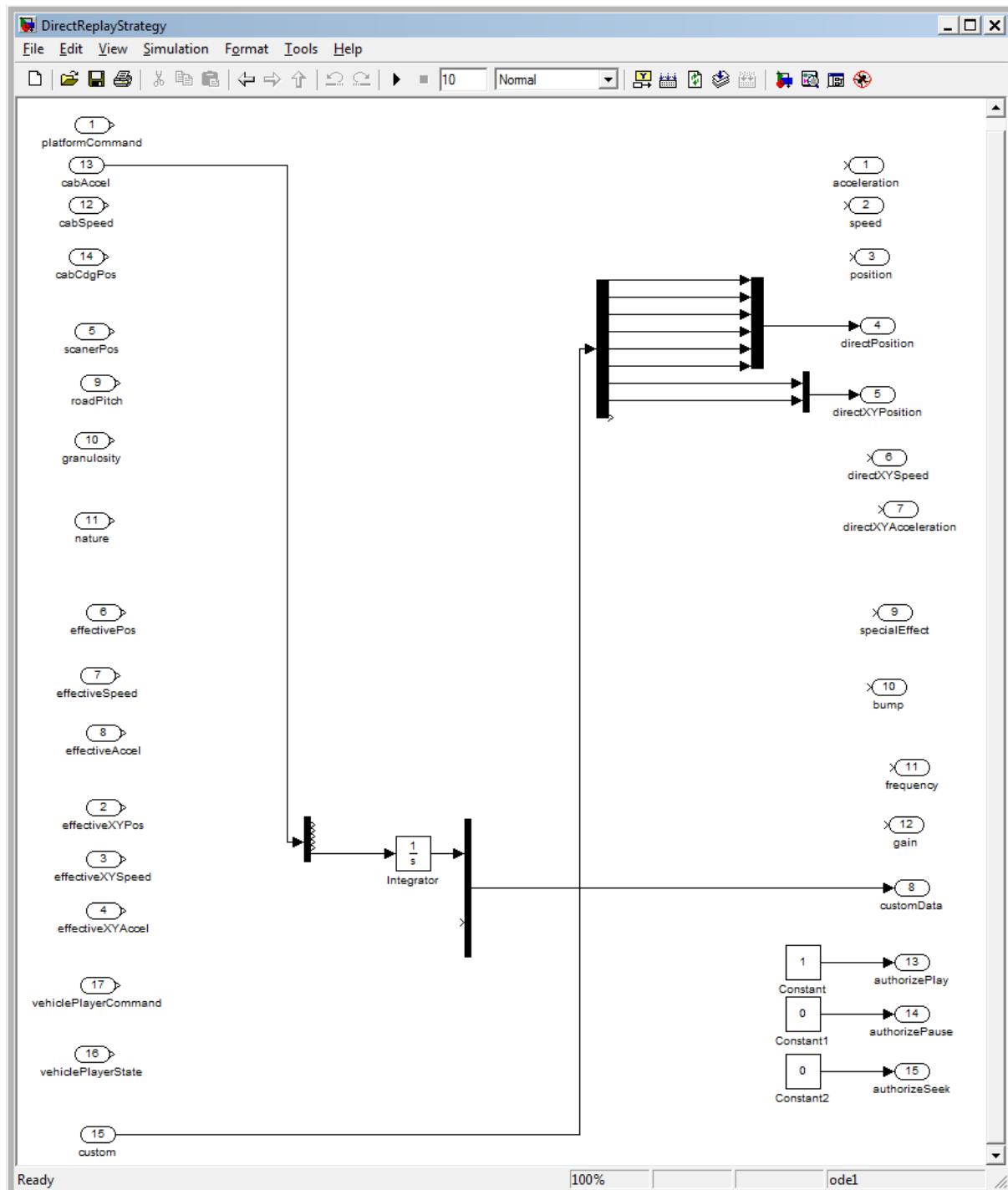


Illustration 104: DirectReplayStrategy Simulink Motion Strategy sample



Do not use this strategy if you have not validated that your vehicle model correctly computes a valid motion platform position in its custom outputs. See chapter about custom vehicle outputs.

To use this strategy, the strategy 8 first custom inputs must be configured to receive the vehicle model 8 first custom outputs. This is configured in the Motion configuration dialogue:

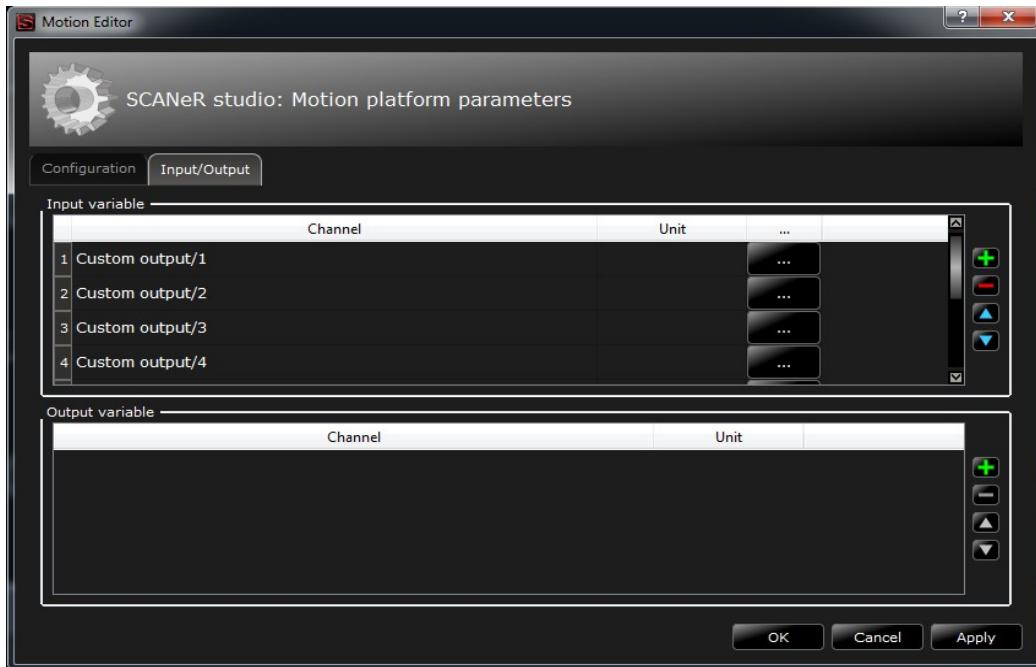


Illustration 105: Motion strategy custom inputs

XIII.B.6 - USING CUSTOM STRATEGIES WITH SCANeR™ STUDIO

Once the custom strategy is compiled, either based on a C sample or a Simulink sample, the strategy will be contained in a DLL. This DLL must be copied to the "<STUDIO_PATH>\<STUDIO_VERS>\bin\<PLATFORM>\plugins" folder of SCANeR™ studio and renamed with a name of the form "Strategy.dll".

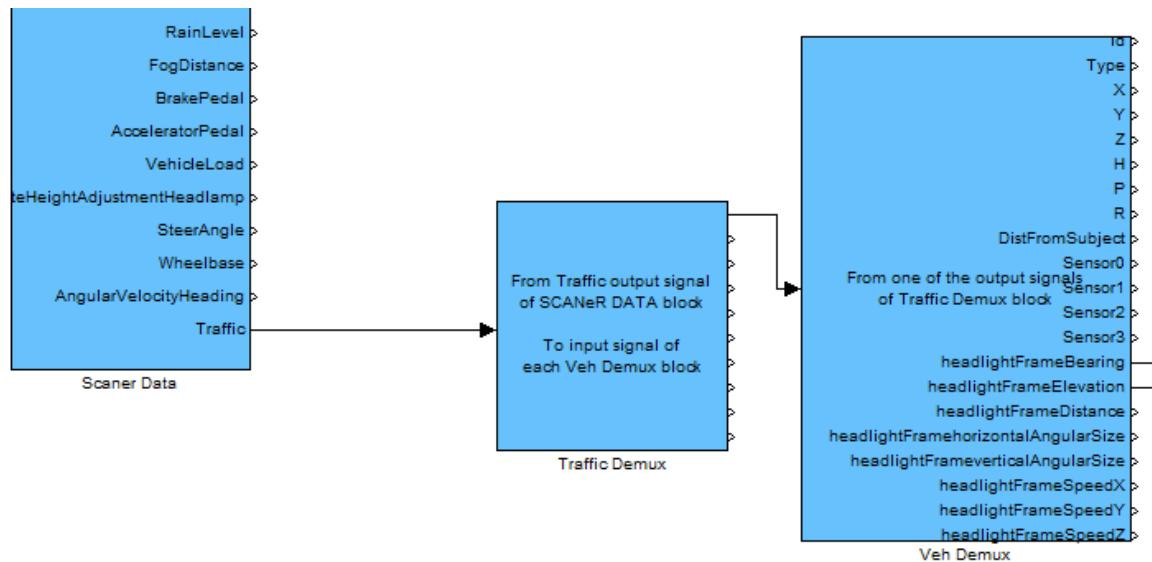


Illustration 106: custom strategy - Simulink

Once the DLL is correctly installed for use, it can be selected in the Motion configuration dialog box. See the Motion configuration chapter.

xiv - AFS INTERFACE

xiv.a - INTRODUCTION

The aim of this document is to describe in detail how to use the AFS Interface Application Programming Interface (API) in order to create your own AFS strategy to use in headlight simulation.

xiv.b - ARCHITECTURE OVERVIEW

To allow several driving simulators to access the AFS functionality without providing an individual solution for each simulator, a general interface has been defined. This interface transfers the vehicle parameters (like simulation time, speed, current curvature, light switch position etc.) from the driving simulator as an input to the AFS-algorithm. It returns the parameters for the individual headlamp pairs like status (on/off), intensity and swivel angles. Of course, since these output parameters will modify the light distribution on the road, they are taken into account for rendering the light distribution into the scene. Therefore, the calculations of these quantities have to be handled in real time (by default 100Hz).

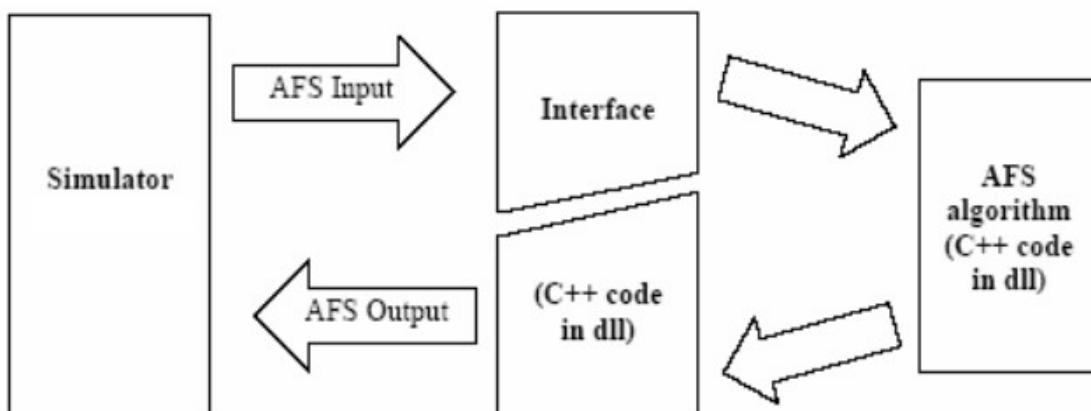


Illustration 107: AFS – Plugin Architecture

Moreover, the algorithms can be tuned with some parameters. It is also possible, to set these parameters via the interface. The way these parameters are set is the same for all algorithms. Hence, when additional algorithms are added, the interface itself doesn't need to change. Additional input parameters for the algorithm don't require modifications of the interface.

Furthermore, it is able to handle a sufficient number of headlamp pairs. On one hand, this is necessary to make sure, that all the headlamps of a vehicle are correctly turned on/off, dimmed, and swivelled. On the other hand, it is desirable to be able to switch between different sets of headlights, to compare for example different AFS strategies, or different types of headlights (e.g. Halogen or Xenon). Last but not least, in order to protect the know how contained in the algorithms, it ensures confidentiality. Internal structures of the algorithms are not disclosed.

This interface has been made in terms of C++ header files containing the declaration of all the interface methods that need to be accessed by the simulator, but no information concerning the inner structure of the algorithms. The DLL includes both, the interface and the algorithms. An additional file can be added to customize the headlamp configuration.

When new algorithms are developed, and need to be integrated, it is accomplished within the DLL. It does not require any modifications in the simulator environment, only the DLL needs to be replaced by a new one. For tuning an algorithm, the interface contains methods, allowing each type algorithm, to:

- check what type of algorithm it is
- what kind of parameters are available
- modify these parameters.

The Headlight Manager (HeadlightMgr) module of SCANeR™ studio software is responsible for the mounting of headlamp inside a vehicle. The positioning information is transmitted to the visual module, through network communication, in order to make the corresponding visual restitution. A user can change the headlamp position and orientation in real time through a GUI included in the HeadlightMgr module. The strategy control interface is carried out and run by the HeadlightMgr. For each time step, output parameters of the interface are used in order to send through the network the new state for each headlamp. For each time step, the HeadlightMgr will call the DLL's main method and ensure the transmission of the updated headlamp information to the visual module.

The following diagram shows the corresponding software architecture:

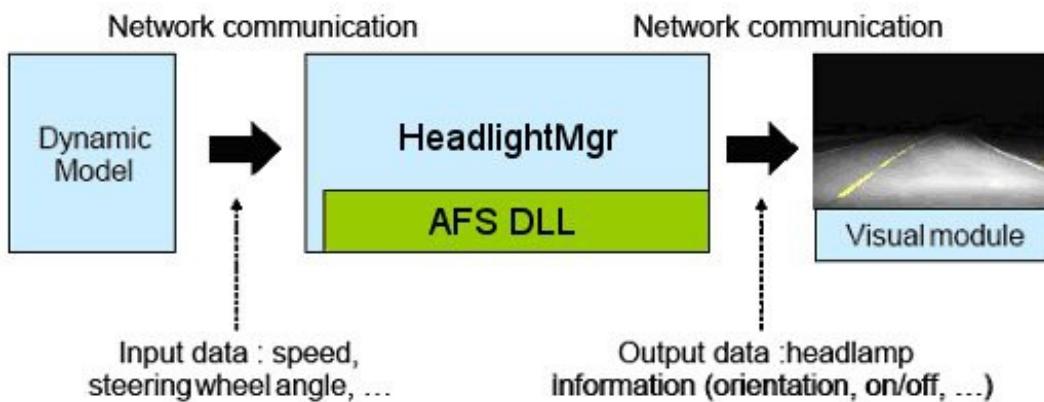


Illustration 108: AFS - Software architecture

The HeadlightMgr integrates a graphic user interface allowing a user to modify in real time various lighting parameters. A specific graphical user interface panel has been designed in order to modify in real time control strategies parameters.

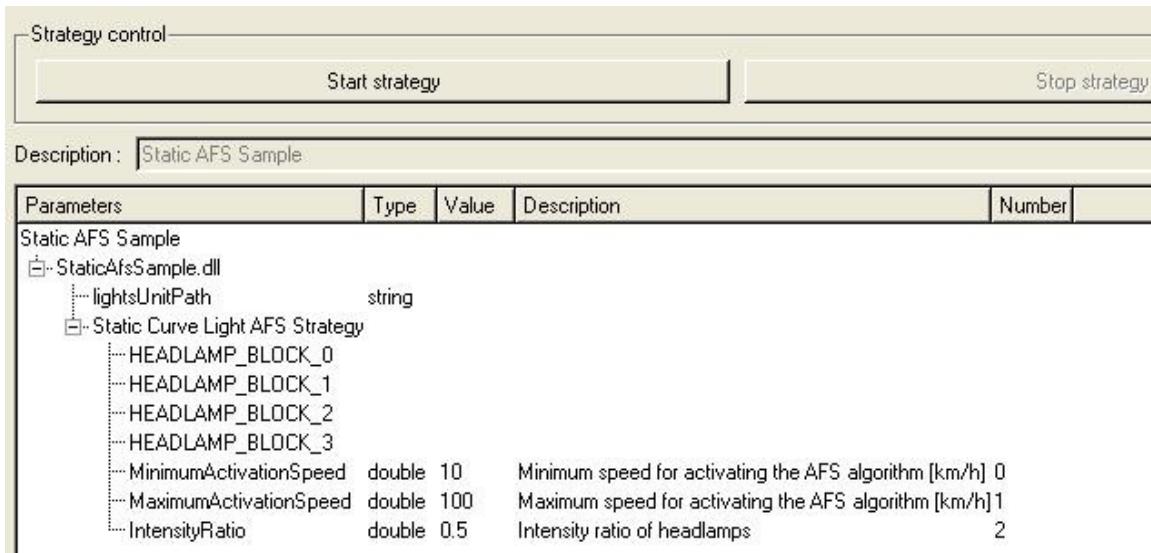


Illustration 109: AFS - Headlight manager GUI

The graphic user interface exposes the following information:

- A list of editable text boxes which represent the list of input parameters for an AFS control strategy.
- A list of non editable text boxes which represent the list of output parameters delivered by an AFS control strategy.

A strategy file (.str) is needed to launch the DLL. It defines the name, the DLL, and the editable input parameters default value.

xiv.c - INPUT/OUTPUT DATA DEFINITIONS

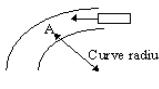
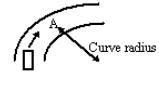
Here is the list of input/output data available for the AFS evaluation. The index of the tables could serve as index entries for input/output parameters look-up tables.

The main information block contains values that are relative to the subject vehicle. There are additional information blocks that contain information about the 10 closest traffic vehicles.

XIV.C.1 - INPUT DATA DEFINITIONS (TO THE DLL)

This paragraph gives the naming convention and associated data format of vehicle data inputs for control strategies. Parameters data format is the C++ double data type.

Nb	Input Name in the C++ Code	Unit	Description
0	Speed	Km/h	
1	SteeringWheelAngle	Degrees	SteeringWheelAngle < 0 if turned to the right SteeringWheelAngle > 0 if turned to the left
2	CurrentCurveCurvature	1/Meters	

Nb	Input Name in the C++ Code	Unit	Description
3	CurveCurvatureAtXMeters	1/Meters	<p>Left turn: Curve radius at point A is set positive</p>  <p>Right turn: Curve radius point A is made negative</p>  <p>The curve curvature is the inverse curve radius value.</p>
4	TheoreticalTime	Seconds	Theoretical time (considering a regular step) passed since the beginning of the strategy
5	MeasuredTime	Seconds	Real measured time passed since the beginning of the strategy
6	X	Meters	Position & orientation of the vehicle in the SCANeR co-ordinate system convention.
7	Y	Meters	
8	Z	Meters	
9	Heading	Degrees	
10	Pitch	Degrees	
11	Roll	Degrees	
12	HeadingRelativeToTheRoad	Degrees	Angle between the vehicle X axis of the vehicle and the direction of the road
13	Reserved for future use		
14	GearBox		-1 for reverse, 0 for neutral, 1, 2, 3, 4, 5
15	Indicators		0 for none, 1 for left, 2 for right
16	Warning		0 for none, 1 for ON
17	LightStatus		0 for none, 1 for POSITION, 2 for DIPPED, 3 for FULL
18	LongitudinalAcceleration	Meters/Second ²	Acceleration of the vehicle on its X axis
19	DistanceToNextVehicleInSameDirection	Meters	Distance D between the front of the vehicle and the back of the next vehicle in the same direction

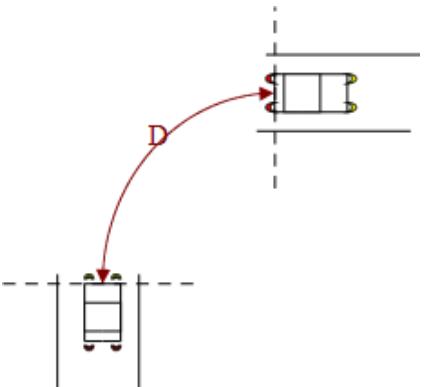
Nb	Input Name in the C++ Code	Unit	Description
			
20	DistanceToNextVehicleInOppositeDirection		
21	SnowLevel		
22	RainLevel		In [0,1] 0 for none, 1 for rainy
23	FogDistance		
24	BrakePedal		In [0,1] 0 for released, 1 for pressed
25	AcceleratorPedal		In [0,1] 0 for released, 1 for pressed
26	VehicleLoad	Kg	Additional load of the vehicle
27	RemoteHeightAdjustmentHeadlamp		Initial pitch angle of the headlamp In [0,1] 0 for 0 degrees, 1 for n degrees.
28	SteerAngle	Degrees	Angle between wheels and vehicle axle
29	Wheelbase	Meters	Distance between rear wheel axle and front wheel axle
30	AngularVelocityHeading	Degrees/Second	Angular Velocity of the rotation of the vehicle around its vertical axis
30	Traffic		Information about the 10 closest traffic vehicles
31	CustomInput		Table of 128 floats containing personalizable data. Can be filled by various processes
32	CustomOutput		Table of 128 floats containing personalizable data. Can be filled by various processes

Table 31 : AFS – Input data

Input value name	Unit	Description
id		Identifier of the traffic vehicle
type		Type (car, bus etc...)
x	m	X coordinate of the vehicle SCANeR frame in the absolute frame
y	m	Y coordinate of the vehicle SCANeR frame in the absolute frame
z	m	Z coordinate of the vehicle SCANeR frame in the absolute frame
heading	rad	Heading of the vehicle
pitch	rad	Pitch of the vehicle
roll	rad	Roll of the vehicle
DistFromSubject	m	Distance from the target vehicle SCANeR frame to the traffic vehicle SCANeR frame
Sensor[4]	lux	Luminance measured by the sensors attached to the traffic vehicle
headlightFrameBearing	rad	Bearing angle of the traffic vehicle center in the target vehicle headlight frame.
headlightFrameElevation	rad	Elevation angle of the traffic vehicle centre in the target vehicle headlight frame.
headlightFrameDistance	m	Distance of the traffic vehicle centre to the subject vehicle headlight frame
headlightFrameHorizontalAngularSize	rad	Horizontal angular size of the traffic vehicle as seen from the subject vehicle headlight frame
headlightFrameVerticalAngularSize	rad	Vertical angular size of the traffic vehicle as seen from the subject vehicle headlight frame

headlightFrameSpeedX	m/s	Relative speed vector of the traffic vehicle in the subject vehicle headlight frame
headlightFrameSpeedY	m/s	
headlightFrameSpeedZ	m/s	

AFS – *input data, traffic vehicle information*

XIV.C.2 - INPUT DATA ORIGINS (TO THE DLL)

This paragraph gives the origin of each input data.

Several input data origins can be identified:

Input data filled through the:

- Simulation messages. Some messages can be directly send as input data but others need some pre-calculations before being interpreted as input data.
- MDL files.
- MICE scenarios.
- Graphical User Interface.

The following table shows the original source of each input value.

Input Name within the C++ Code	Class Message	Name	Field	Comment
Speed	Msg_class_vehicle	vehicleUpdate	speed	
SteeringWheelAngle	Msg_class_model	ModellInput	steeringWheel	Multiplied by a wheel coefficient coming from the GUI
CurrentCurveCurvature				Exported from MICE scenario
CurveCurvatureAtXMeters				Exported from MICE scenario
TheoreticalTime				Calculated from the GUI
MeasuredTime				Measured from the GUI
X	Msg_class_vehicle	vehicleUpdate	pos[0]	
Y	Msg_class_vehicle	vehicleUpdate	pos[1]	
Z	Msg_class_vehicle	vehicleUpdate	pos[2]	
Heading	msg_class_vehicle	vehicleUpdate	pos[3]	
Pitch	msg_class_vehicle	vehicleUpdate	pos[4]	
Roll	msg_class_vehicle	vehicleUpdate	pos[5]	

Input Name within the C++ Code	Class Message	Name	Field	Comment
HeadingRelativeToTheRoad				Exported from MICE scenario
GeographicClassification				
GearBox	msg_class_model	ModellInput	gearbox	
GearBoxAutoMode		ModellInput	gearboxAutoMode	
Indicators	msg_class_model	ModellInput	indicators	
Warning	msg_class_model	ModellInput	warnings	
LightStatus	msg_class_model	ModellInput	lights	
LongitudinalAcceleration	msg_class_model	vehicleUpdate	accel[0]	
DistanceToNextVehicleInSameDirection				Exported from MICE scenario
DistanceToNextVehicleInOppositeDirection				
SnowLevel				
RainLevel				Exported from MICE scenario
FogDistance				
BrakePedal	msg_class_model	ModellInput	brake	
AcceleratorPedal	msg_class_model	ModellInput	accelerator	
VehicleLoad				Exported from MDL files
RemoteHeightAdjustmentHeadlamp				Exported from the GUI (will be added to the ModellInput part of the msg_class_model).
SteerAngle	msg_class_model	ModellInput	steeringWheel	Multiplied by max steer angle exported from MDL files
Wheelbase				Exported from MDL files
AngularVelocityHeading	msg_class_model	vehicleUpdate	speed[3]	
Traffic	Msg_class_vehicle	vehicleUpdate		

Table 32 : AFS - origins of input data

For input data coming from MICE, you need to use the following data/ExportChannel association :

- CurrentCurveCurvature = ExportChannel 0
- CurveRadiusAt10Meters = ExportChannel 1
- HeadingRelativeToTheRoad = ExportChannel 2

- DistanceToNextVehicleInSameDirection = ExportChannel 5
- RainLevel = ExportChannel 6

XIV.C.3 - OUTPUT DATA DEFINITIONS (FROM THE DLL)

Each headlampBlock contains 2 headlamps carrying position data (heading, pitch and roll fields) and voltage data (intensity field). The status field indicates when the headlamp is activated or not.

From the output of the AFS DLL, the following data structures are retrieved :

Name	Field	Comment
Headlamp	bool status	on / off
	double heading	angle in degrees
	double pitch	angle in degrees
	double roll	angle in degrees
	double intensity	voltage data
HeadlampBlock	char comment[1024]	comment about HeadlampBlock
	Headlamp leftHeadlamp	left headlamp of block
	Headlamp rightHeadlamp	right headlamp of block

Table 33 : AFS – Output data structures

XIV.C.4 - INPUT PARAMETERS DEFINITION

Input parameters are designed to give a certain amount of control on the behaviour of a headlamp strategy to the final user. In order to have the possibility to handle several static tables with only one DLL, in case of complex lighting strategies, an input parameter will represent the name of a static table file. According to this statement, the following data organisation for input parameters is the following:

Name	Field	Comment
DoubleParameter	string name	name of parameter
	string description	purpose of this parameter
	double data	double value of parameter
StringParameter	string name	name of parameter
	string description	purpose of this parameter
	string data	string value of parameter

Table 34 : AFS – Input data structures

Within this framework, it is expected that the DLL will be able to return all parameters with the double types as well as all parameters with the string type. A dedicated GUI manages these informations for the

preparation of a lighting experiment. Input parameters entries are displayed with their names. The description field is used as information for the experimenter.

XIV.D - DEVELOPER GUIDE

XIV.D.1 - OVERVIEW

The Application Programming Interface (API) is provided as a DLL with C++ interface.

The following chapters present how to use this API and, in addition, samples are provided with the API.

The content of the delivering directory is as follow:

- DOC: contains this documentation
- INCLUDE: contains C++ header files
- SAMPLES: contains 3 AFS strategies examples
- DATA: contains headlight data files to use with the examples
- SIMULINK: contains sample projects for MatlabSimulink

XIV.D.2 - DEFINITION OF THE DLL INTERFACE

An interface has been defined to allow the SCANeR™ studio HeadlightManager module to dynamically load the DLLs.

Here are the different functions with their parameters.

Function name	getNumberOfHeadlampsBlocks	
Description	Returns the total number of headlamp blocks	
Prototype	int getNumberOfHeadlampsBlocks()	
Input value(s)	void	
Return value	int	total number of headlamp blocks (maximum 12)
Function name	run	
Description	Updates all headlamp blocks	
Prototype	void run(double* inputData)	
Input value(s)	double* inputData	table of input data
Return value	void	
Function name	getHeadlampsBlock	
Description	Updates and returns the headlamp block	
Prototype	const HeadlampsBlock& getHeadlampsBlock(int headLampsBlock, double* inputData)	
Input value(s)	int headLampsBlock	the corresponding headlampsBlock
	double* inputData	table of input data
Return value	const HeadlampsBlock&	the updated headlampsBlock
Function name	getHeadlampsBlock	
Description	Updates and returns the headlamp block	
Prototype	const HeadlampsBlock& getHeadlampsBlock(char* comment, double* inputData)	
Input value(s)	char* comment	the name field of the headlamp block
	double* inputData	table of input data
Return value	const HeadlampsBlock&	the updated headlampsBlock
Function name	getHeadlampsBlock	
Description	Returns the headlamp block without updating it	
Prototype	const HeadlampsBlock& getHeadlampsBlock(int headLampsBlock)	
Input value(s)	int headLampsBlock	the corresponding headlampsBlock
Return value	const HeadlampsBlock&	the non updated headlampsBlock
Function name	getHeadlampsBlock	
Description	Returns the headlamp block without updating it	
Prototype	const HeadlampsBlock& getHeadlampsBlock(char* comment)	
Input value(s)	char* comment	the name field of the headlamp block
Return value	const HeadlampsBlock&	the non updated headlampsBlock

Function name	getNumberOfDoubleParameters	
Description	Returns the number of double parameters for the headlampBlock	
Prototype	int getNumberOfDoubleParameters(int headLampBlock)	
Input value(s)	int headLampBlock	the corresponding headlampBlock
Return value	int	the number of double parameters
Function name	getDoubleParameter	
Description	Returns the corresponding double parameter for the specified headlampBlock	
Prototype	DoubleParameter getDoubleParameter(int headLampBlock, int parameter)	
Input value(s)	int headLampBlock	the corresponding headlampBlock
	int parameter	the corresponding parameter
Return value	DoubleParameter	the double parameter
Function name	setDoubleParameter	
Description	Sets the corresponding double parameter for the specified headlampBlock to newValue	
Prototype	int setDoubleParameter(int headLampBlock, int parameter, double newValue)	
Input value(s)	int headLampBlock	the corresponding headlampBlock
	int parameter	the corresponding parameter
	double newValue	the new value of the double parameter
Return value	int	return 0 in case of success
Function name	getNumberOfStringParameters	
Description	Returns the number of string parameters for the given headlampBlock	
Prototype	int getNumberOfStringParameters(int headLampBlock)	
Input value(s)	int headLampBlock	the corresponding headlampBlock
Return value	int	the number of string parameters
Function name	getStringParameter	
Description	Returns the corresponding string parameter for the specified headlampBlock	
Prototype	StringParameter getStringParameter(int headLampBlock, int parameter)	
Input value(s)	int headLampBlock	the corresponding headlampBlock
	int parameter	the corresponding parameter
Return value	StringParameter	the string parameter
Function name	setStringParameter	
Description	Sets the corresponding string parameter for the specified headlampBlock to newValue	
Prototype	int setStringParameter(int headLampBlock, int parameter, char* newValue)	

<i>Input value(s)</i>	int headLampBlock	the corresponding headlampBlock
	int parameter	the corresponding parameter
	char* newValue	the new value of the string parameter
<i>Return value</i>	int	return 0 in case of success
Function name	setAlgorithmDoubleParameter	
<i>Description</i>	Sets the corresponding double parameter specified by its path name to newValue	
<i>Prototype</i>	int setAlgorithmDoubleParameter(char* commentParam, double value)	
<i>Input value(s)¹²</i>	char* commentParam	the path name of the double parameter
	double newValue	the new value of the double parameter
<i>Return value</i>	int	return 0 in case of success
Function name	setAlgorithmStringParameter	
<i>Description</i>	Sets the corresponding string parameter specified by its path name to newValue	
<i>Prototype</i>	int setAlgorithmStringParameter(char* commentParam, char* value)	
<i>Input value(s)</i>	char* commentParam	the path name of the string parameter
	char* newValue	the new value of the string parameter
<i>Return value</i>	int	return 0 in case of success
Function name	getAlgorithmDoubleParameter	
<i>Description</i>	Returns the corresponding double parameter for the specified path name of the double parameter	
<i>Prototype</i>	DoubleParameter getAlgorithmDoubleParameter(char* commentParam)	
<i>Input value(s)</i>	char* commentParam	the path name of the double parameter
<i>Return value</i>	DoubleParameter	the corresponding double parameter
Function name	getAlgorithmStringParameter	
<i>Description</i>	Returns the corresponding string parameter for the specified path name of the string parameter	
<i>Prototype</i>	StringParameter getAlgorithmStringParameter(char* commentParam)	
<i>Input value(s)¹³</i>	char* commentParam	the path name of the string parameter
<i>Return value</i>	StringParameter	the corresponding string parameter
Function name	getAlgorithmName	

12 The string path names (commentParam) double and correspond to the parameter name preceded by the algorithm name(s) separated by “ / ” characters (for instance “strategy1/sub-strategy1/minActivationSpeed”).

13 The string path names (commentParam) correspond to the parameter name preceded by the algorithm name(s) separated by “ / ” characters (for instance “strategy1/sub-strategy1/minActivationSpeed”).

<i>Description</i>	Returns the the corresponding algorithm name for the given headlampBlock											
<i>Prototype</i>	const char* getAlgorithmName(int headLampBlock)											
<i>Input value(s)</i>	int headLampBlock	the headlampBlock number										
<i>Return value</i>	const char*	the corresponding algorithm name										
Function name	getNumberOfRequiredInputData											
<i>Description</i>	Returns number of required input data											
<i>Prototype</i>	int getNumberOfRequiredInputData()											
<i>Input value(s)</i>	void											
<i>Return value</i>	int	the number of required input data										
Function name	getInputDataName											
<i>Description</i>	Returns the name of the input data corresponding to the input index											
<i>Prototype</i>	const char* getInputDataName(int dataIndex)											
<i>Input value(s)</i>	int dataIndex	the index of the data										
<i>Return value</i>	const char*	name of the corresponding input data										
Function name	getLastErrorCode											
<i>Description</i>	Returns the last error code											
<i>Prototype</i>	int getLastErrorCode()											
<i>Input value(s)</i>	void											
<i>Return value</i>	int	<p>error code :</p> <table border="1"> <tr> <td>0</td><td>everything is OK</td></tr> <tr> <td>1</td><td>no headlamp blocks found: empty file or incorrect path</td></tr> <tr> <td>2</td><td>headlamp block index exceeds – 1</td></tr> <tr> <td>3</td><td>attempted to access AFS-parameters for a headlamp block that does not have an AFS-algorithm</td></tr> <tr> <td>4</td><td>failed to decompose string into comment/parameterName</td></tr> </table>	0	everything is OK	1	no headlamp blocks found: empty file or incorrect path	2	headlamp block index exceeds – 1	3	attempted to access AFS-parameters for a headlamp block that does not have an AFS-algorithm	4	failed to decompose string into comment/parameterName
0	everything is OK											
1	no headlamp blocks found: empty file or incorrect path											
2	headlamp block index exceeds – 1											
3	attempted to access AFS-parameters for a headlamp block that does not have an AFS-algorithm											
4	failed to decompose string into comment/parameterName											
Function name	getHeadlampBlockFunction											
<i>Description</i>	Returns the type of the headlamp block corresponding to the input headlampBlock number. It's used to do the mapping between the headlamp block list and the lamp list in the headlight manager.											
<i>Prototype</i>	const char* getHeadlampBlockFunction(int headlampBlock)											
<i>Input value(s)</i>	int headlampBlock	the headlamp block number										

<i>Return value</i>	const char*	the type of the headlamp block. It must be one of these values:
	C	Low beam
	R	High beam
	AB	Fog beam
	ADD1	Additional beam
	C2	Low beam 2
	C3	Low beam 3
	R2	High beam 2
	R3	High beam 3
	AB2	Fog beam 2
	AB3	Fog beam 3
	ADD2	Additional beam 2
	ADD3	Additional beam 3
<i>Function name</i>	createAFS	
<i>Description</i>	Returns the Base AFS Interface object corresponding to the path of the parameter file	
<i>Prototype</i>	BaseAFSInterface* createAFS(char* lightUnitsPath)	
<i>Input value(s)</i>	char* lightUnitsPath	the path of the optional parameter file
<i>Return value</i>	BaseAFSInterface*	the corresponding Base AFS Interface object

Table 35: AFS - Functions description

XIV.D.3 - EXAMPLES

Three examples are provided:

<STUDIO_PATH>\<STUDIO_VERS>\APIs\AFS_API\samples\StaticAfsSample.cpp: a simple static curve light strategy example using the steering wheel angle to change additional headlamps intensity.

<STUDIO_PATH>\<STUDIO_VERS>\APIs\AFS_API\samples\DynamicAfsSample.cpp: a simple dynamic curve light strategy example using the steering wheel angle to change additional headlamps heading.

<STUDIO_PATH>\<STUDIO_VERS>\APIs\AFS_API\samples\FullAfsSample.cpp: a Full AFS strategy example using the speed to switch between 2 cartographies by blending their intensity.

XIV.D.4 - PLATFORM

The AFS API runs on computer described in the "Requirements/Software" chapter of the SCANeRstudio_STUDIO_UserManual documentation.

XIV.D.5 - LICENSE MANAGEMENT

AFS Interface requires a software license as well as SCANeR™ studio module.

The license has to be set in the license configuration file of the selected SCANeR™ studio configuration.

XIV.D.6 - AFS API FOR SIMULINK

XIV.D.6.i - Package

The AFS API for Simulink package includes an AFS_API/Simulink directory which contains a Grt_scanner and SCANER_AFS directories (software components that enable to build a SCANeR™ studio software from a Simulink model) and sample projects for MatlabSimulink.

XIV.D.6.ii - Glossary

Control strategy: a control strategy computes the angles (heading, pitch, roll) and the intensity of an lighting entity. A control strategy is later assigned to projectors and light sources.

Activation strategy: the activation strategy computes the status (on/off) of the different optical types (LowBeam, HighBeam, MotorWay...).

XIV.D.6.iii - Principles

The aim of the AFS API is to compute at each simulation step the angles and intensity of the projectors and light sources.

This computation action is performed in Simulink models, like in the one below:

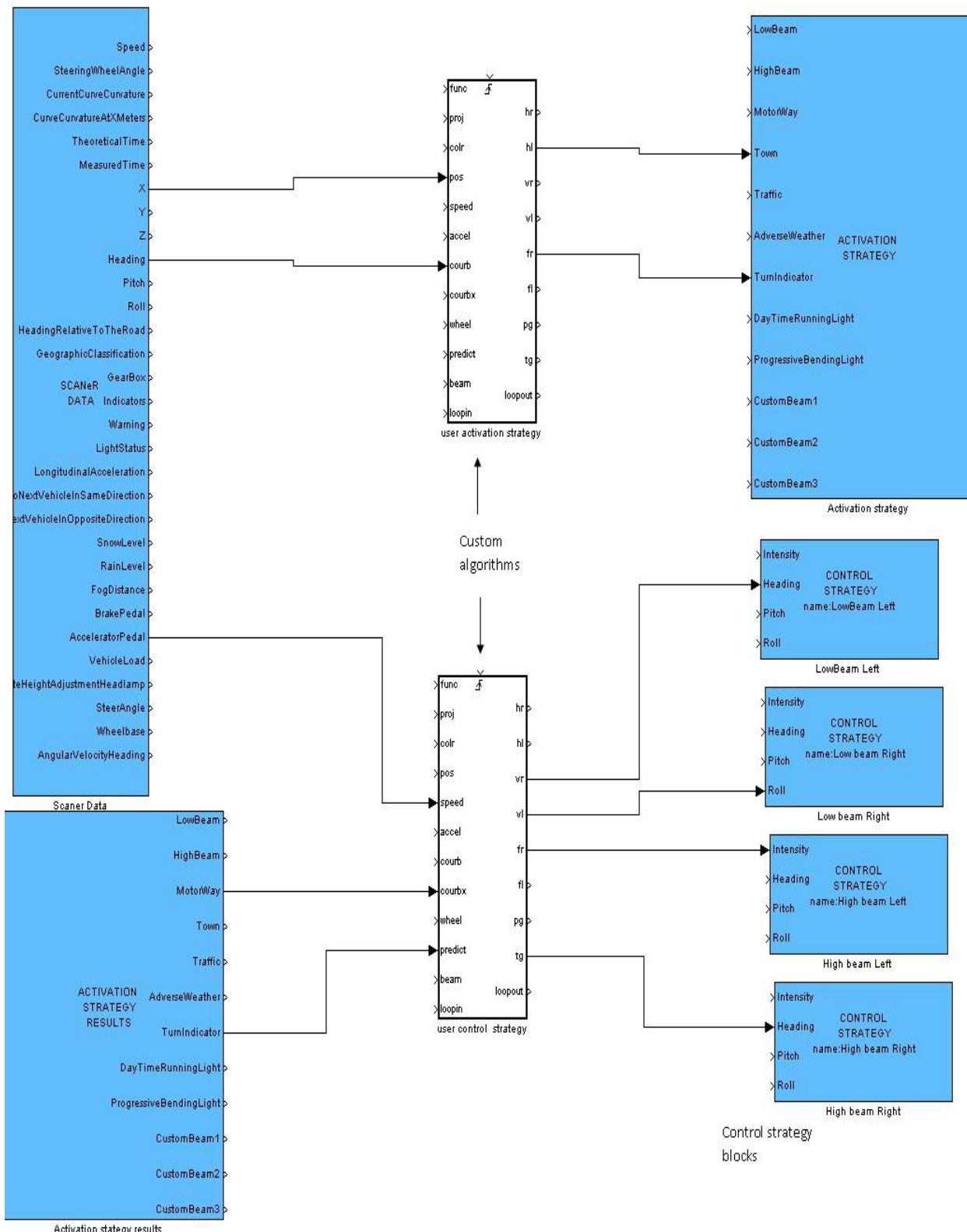


Illustration 110: AFS - Simulink

Example of an AFS Simulink model

All the strategies can be done in one Simulink model, like in this example, or parted in several Simulink models (for instance one Simulink model per optical function).

Below is a brief description of the SCANeR_AFS simulink blocks:

- **SCANeRData** Block (input): provides with all data from SCANeR™ simulation (vehicle speed, road curvature, steering wheel angles, etc...). See complete documentation in section 7.1.2.
- **ControlStrategy** Block (output): sets the angles (heading, pitch, roll) and the intensity of an entity. This block is later assigned (outside Simulink) to the projectors and light sources in the NightTestManager software.
- **ActivationStrategy** Block (output): the activation strategy computes the status (on/off) of the different optical types.
- **ActivationStrategyResults** Block (input): this block is a copy of the values set in the **ActivationStrategy** Block.

The activation strategy has no real effect on the intensity of the projectors and light sources. It is input information that a control strategy block uses to compute its intensity.

When a projector/light source is controlled by a Simulink control strategy block, its lighting status is only decided by the value of the intensity parameter of the control strategy block, not by the activation strategy block.

To be used in SCANeR™, the Simulink model is computed in an executable file¹⁴.

The angles values of a control strategy block are either assigned to a projector or a light source. The figure below shows what the angles of a control strategy block represent in both cases (assigned to a light source or a projector):

¹⁴ 32bits version. Read more about Simulink in the "Environment" chapter (page 18).

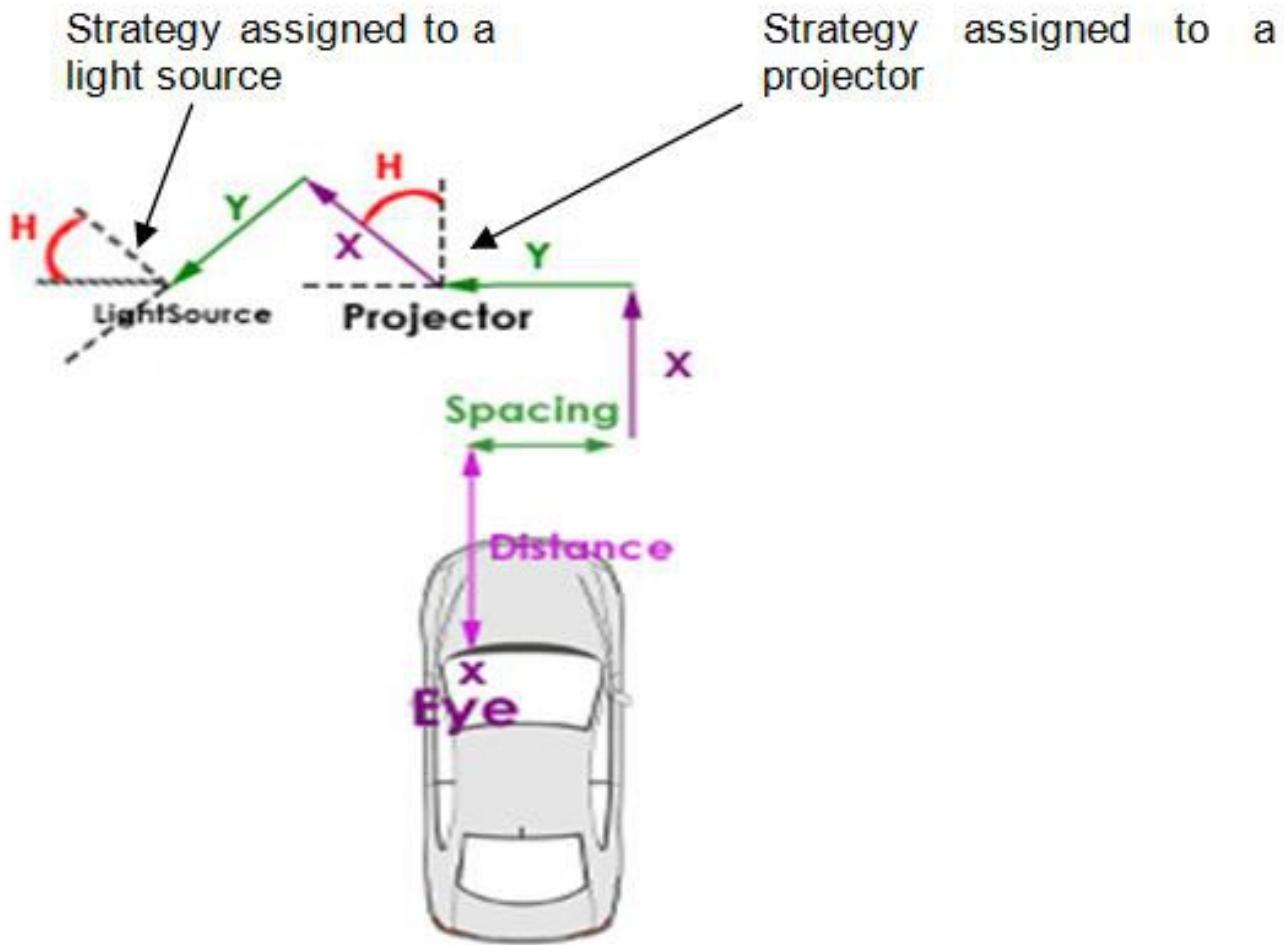


Illustration 111 : AFS – projector light sources

Installation

A working SCANeR™ studio execution environment must be installed.

- Environment variable:

Create an environment variable `SCANER_AFS=<Path to the AFS_API/Simulink folder>`

- MATLAB Pathes

Add following folders to the MATLAB Path (Menu File/Set Path...):

`$SCANER_AFS`

`$SCANER_AFS\grt_scanner`

`$SCANER_AFS\SCANeR_AFS`

- Simulink Library

The Simulink Library Browser should contain the SCANeR™ AFS Library containing blocks for creating AFS Strategy.

XIV.D.6.iv - AFS strategy edition

XIV.D.6.iv.a - In Matlab Simulink

XIV.D.6.iv.a.1 - Step 1 : Opening Matlab

- Launch Matlab
- Set Matlab current directory where the Simulink model (.mdl file) will be saved.

XIV.D.6.iv.a.2 - Step 2 : Editing the Simulink Model

- Start from an existing model, or from a new one. The useful blocks can be found in the Simulink library browser, in the SCANeR™ studio AFS directory.
- Edit the model as explained in section 5 Principles.

Here is a complete description of the SCANeR™ AFS blocks:

SCANeRData Block (input): provides with all data from SCANeR™ studio simulation (vehicle speed, road curvature, steering wheel angles, etc...).

PARAMETER NAME	UNIT	VALUE / DESCRIPTION
Speed In bl	Km/h	
SteeringWheelAngle	-1/1	SteeringWheelAngle < 0 if turned to the right SteeringWheelAngle > 0 if turned to the left
CurrentCurveCurvature	1/Meters	The curve curvature is the inverse curve radius value.
CurveCurvatureAtXMeters	1/Meters	
TheoreticalTime	Seconds	Theoretical time (considering a regular step) passed since the beginning of the strategy
MeasuredTime	Seconds	Real measured time passed since the beginning of the strategy
X	Meters	Position & orientation of the vehicle in the RENAULT co-ordinate system convention.
Y	Meters	
Z	Meters	
Heading	Degrees	
Pitch	Degrees	
Roll	Degrees	
HeadingRelativeToTheRoad	Degrees	Angle between the vehicle X axis of the vehicle and the direction of the road
GeographicClassification		-1 for unknown, 0 for city, 1 for country, 2 for motorway, 3 for mountain, 4 for tunnel
GearBox		-1 for reverse, 0 for neutral, 1, 2, 3, 4, 5
Indicators		0 for none, 1 for left, 2 for right
Warning		0 for none, 1 for ON
LightStatus		0 for none, 1 for POSITION, 2 for DIPPED, 3 for FULL
LongitudinalAcceleration	Meters / Second2	Acceleration of the vehicle on its X axis
DistanceToNextVehicleInSameDirection	Meters	Distance D between the front of the vehicle and the back of the next vehicle in the same direction

PARAMETER NAME	UNIT	VALUE / DESCRIPTION
DistanceToNextVehicleInOppositeDirection	Meters	Distance between the front of the vehicle and the front of the next vehicle in the opposite direction
SnowLevel		In [0,1] 0 for none, 1 for snowy
RainLevel		In [0,1] 0 for none, 1 for rainy
FogDistance	Meters	Visibility distance in fog conditions
BrakePedal		In [0,1] 0 for released, 1 for pressed
AcceleratorPedal		In [0,1] 0 for released, 1 for pressed
VehicleLoad	Kg	Additional load of the vehicle
RemoteHeightAdjustmentHeadlamp		Initial pitch angle of the headlamp In [0,1] 0 for 0 degrees, 1 for n degrees.
SteerAngle	Degrees	Angle between wheels and vehicle axle
Wheelbase	Meters	Distance between rear wheel axle and front wheel axle
AngularVelocityHeading	Degrees / Second	Angular Velocity of the rotation of the vehicle around its vertical axis
Traffic		Informations about the 10 closest traffic vehicles. Sorted from nearest to farest.

Tableau 36: AFS – SCANeRData Block - Input

For each detected traffic vehicle, following informations are available by using the demux bloc.

Input value name	Unit	Description
<code>id</code>		Identifier of the traffic vehicle. (-1 if not present)
<code>type</code>		Type (car, bus etc...)
<code>x</code>	m	X coordinate of the vehicle SCANeR frame in the absolute frame
<code>y</code>	m	Y coordinate of the vehicle SCANeR frame in the absolute frame
<code>z</code>	m	Z coordinate of the vehicle SCANeR frame in the absolute frame
<code>heading</code>	rad	Heading of the vehicle
<code>pitch</code>	rad	Pitch of the vehicle
<code>roll</code>	rad	Roll of the vehicle
<code>distFromSubject</code>	m	Distance from the target vehicle SCANeR frame to the traffic vehicle SCANeR frame
<code>sensor[4]</code>	lux	Luminance measured by the 4 first headlight sensors attached to this traffic vehicle.
<code>headlightFrameBearing</code>	rad	Bearing (horizontal) angle of the traffic vehicle center in the target vehicle headlight frame.
<code>headlightFrameElevation</code>	rad	Elevation (vertical) angle of the traffic vehicle centre in the target vehicle headlight frame.
<code>headlightFrameDistance</code>	m	Distance of the traffic vehicle centre to the subject vehicle headlight frame
<code>headlightFrameHorizontalAngularSize</code>	rad	Horizontal angular size of the traffic vehicle as seen from the subject vehicle headlight frame
<code>headlightFrameVerticalAngularSize</code>	rad	Vertical angular size of the traffic vehicle as seen from the subject vehicle headlight frame
<code>headlightFrameSpeedX</code>	m/s	X coordinate of the relative speed vector of the traffic vehicle in the subject vehicle headlight frame
<code>headlightFrameSpeedY</code>	m/s	Y coordinate of the relative speed vector of the traffic vehicle in the subject vehicle headlight frame
<code>headlightFrameSpeedZ</code>	m/s	Z coordinate of the relative speed vector of the traffic vehicle in the subject vehicle headlight frame

		vehicle in the subject vehicle headlight frame
--	--	--

AFS – input data, traffic vehicle informations**Example of processing such infomations:**

Computation of ABS(Subject Heading – (Traffic vehicle heading * 180/PI)) give the relative orientation:

If < 90: Vehicleq in the same direction

If > 90: Vehicles in opposite direction

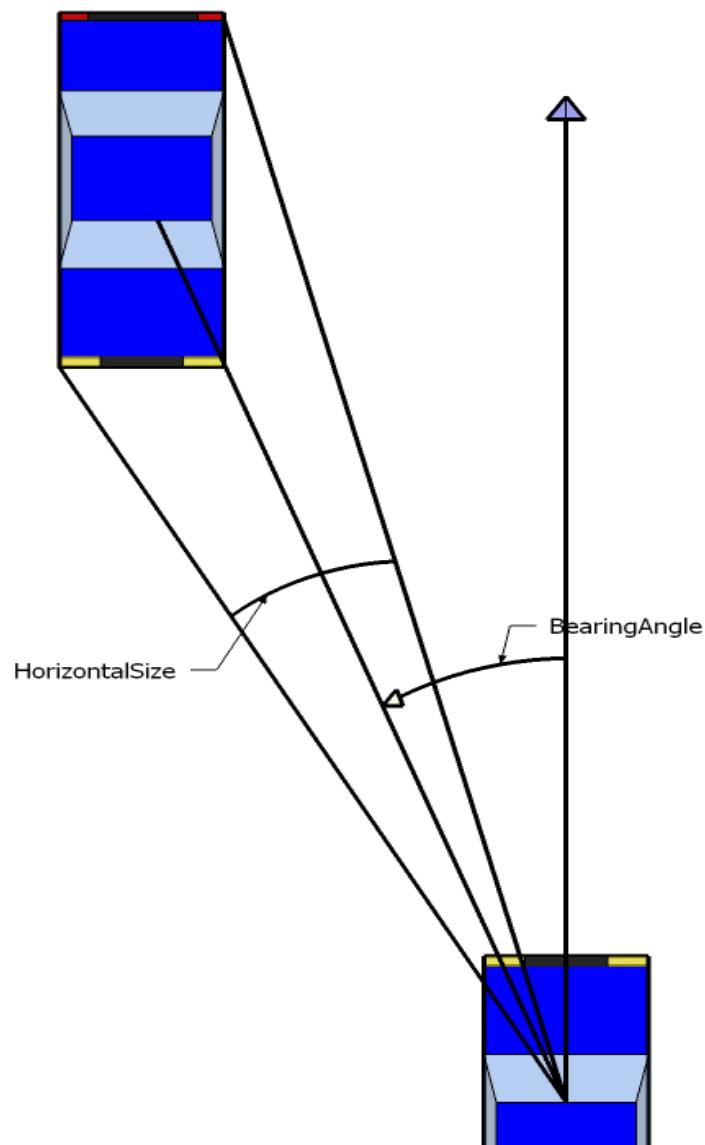


Illustration 112:

ControlStrategy Block (output): sets the angles (heading, pitch, roll) and the intensity of a light entity.

PARAMETER NAME	UNIT	VALUE / DESCRIPTION
Intensity	factor	Gain applied on the nominal intensity
Heading	Degrees	Rotation vector is towards up
Pitch	Degrees	Rotation vector is towards right
Roll	Degrees	Rotation vector is towards front

Tableau 37: AFS – ControlStrategy Block - Output

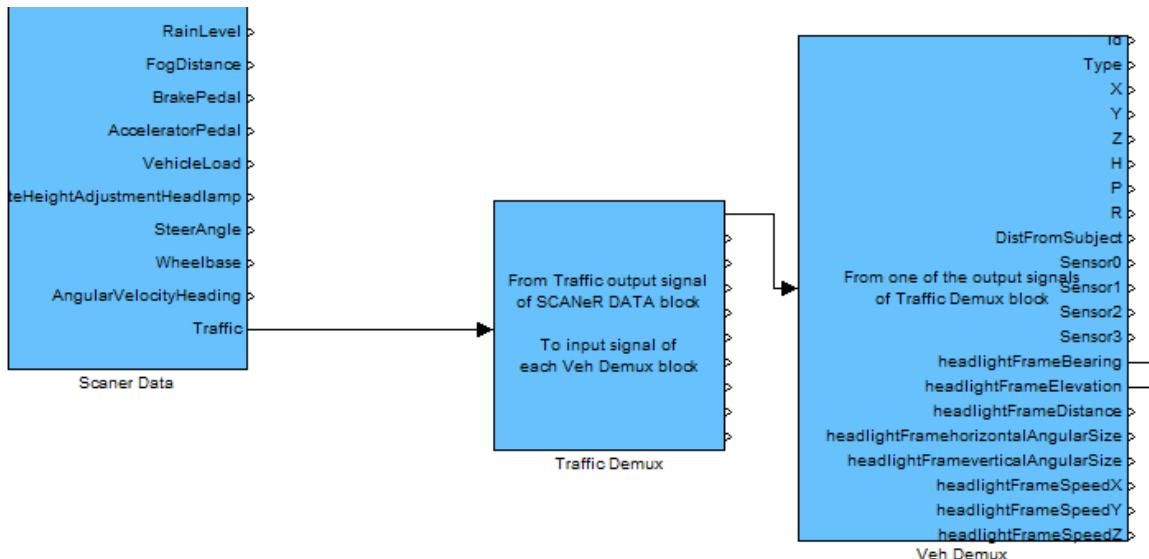
ActivationStrategy Block (output): the activation strategy computes the status (on/off) of the different optical types.

PARAMETER NAME	UNIT	VALUE / DESCRIPTION
LowBeam	0 / 1	On/off activation
HighBeam	0 / 1	On/off activation
MotorWay	0 / 1	On/off activation
Town	0 / 1	On/off activation
Traffic	0 / 1	On/off activation
AdverseWeather	0 / 1	On/off activation
TurnIndicator	0 / 1	On/off activation
DayTimeRunningLight	0 / 1	On/off activation
ProgressiveBendingLight	0 / 1	On/off activation
CustomBeam1	0 / 1	On/off activation
CustomBeam2	0 / 1	On/off activation
CustomBeam3	0 / 1	On/off activation

Tableau 38: AFS - ActivationStrategy Block - Output

ActivationStrategyResults Block (input): this block is a copy of the values set in the ActivationStrategy Block.

Traffic Demux and **Veh Demux** block



By plugging the **Traffic Demux** block on to the **SCANeR Data** Traffic output, the Simulink model can access data of the 10 closest traffic vehicles to the target vehicle. The Veh Demux must be used to extract individual values for each traffic vehicle.

See the traffic vehicle input table for the meaning of each of these values.

XIV.D.6.iv.a.3 - Step 3 : Generating the executable file

- Make sure the RTW generation target is set to `grt_scanner.tlc` OKTAL AFS SCANeR™ Real Time target (Simulink menu Tools/Real Time Workshop/Options)
- Generate the executable (simulink menu Tools/Real Time Workshop/Build model or CTRL+B shift)

The executable file generates in the same directory as the Simulink model file. If not, modify the Matlab current directory



Remark: you can also choose the “`grt_scanner.tlc` Visual C++ project makefile” target to generate source code in Visual C++ project format instead of an executable.

XIV.D.6.iv.a.4 - Step 4 : Copying the executable file In the data directory

To do this, you can use the command line Matlab instruction `copyfile('executable path and name','Data/Headlights/Strategy directory')`

XIV.D.6.iv.b - In NightTestManager Software

The AFSManager software must be launched in the same time as the NightTestManager software.

1. Select a product, and choose a simulink executable containing an activation strategy

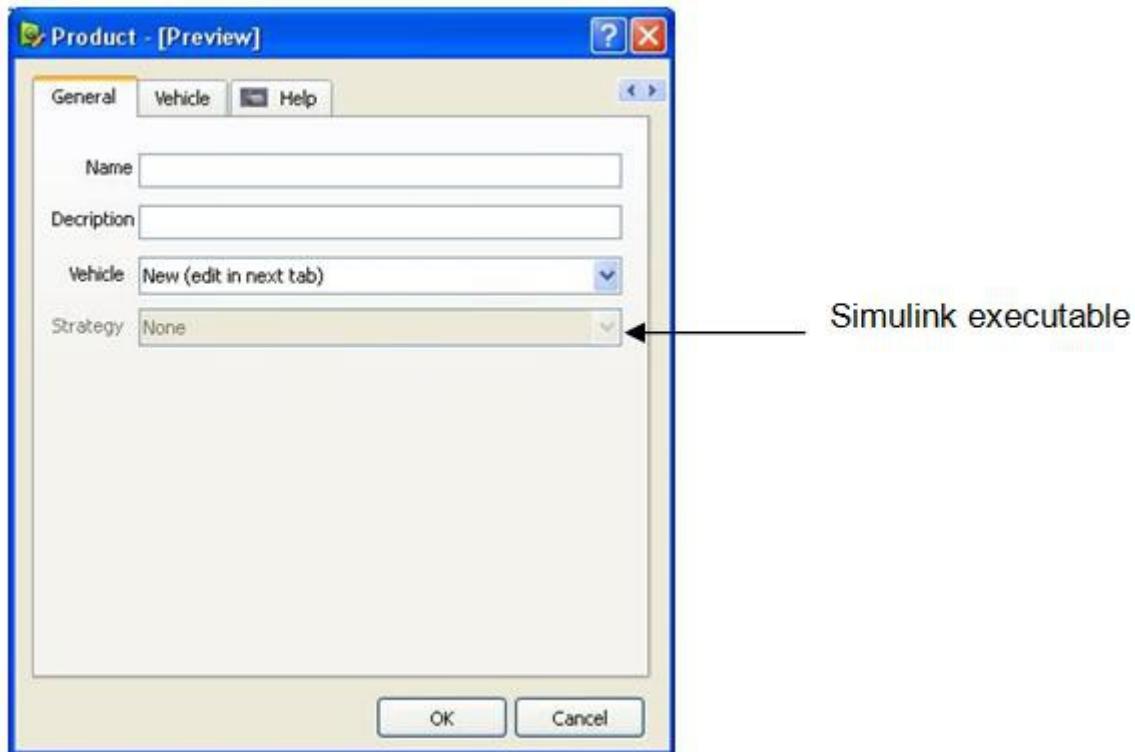


Illustration 113: NightTestManager Software - simulink executable

This action is optional, because the activation strategy is optional, and because the activation strategy can also be contained in the simulink executable selected in the optical functions (sees below)

2. Select an optical function, and choose a simulink executable

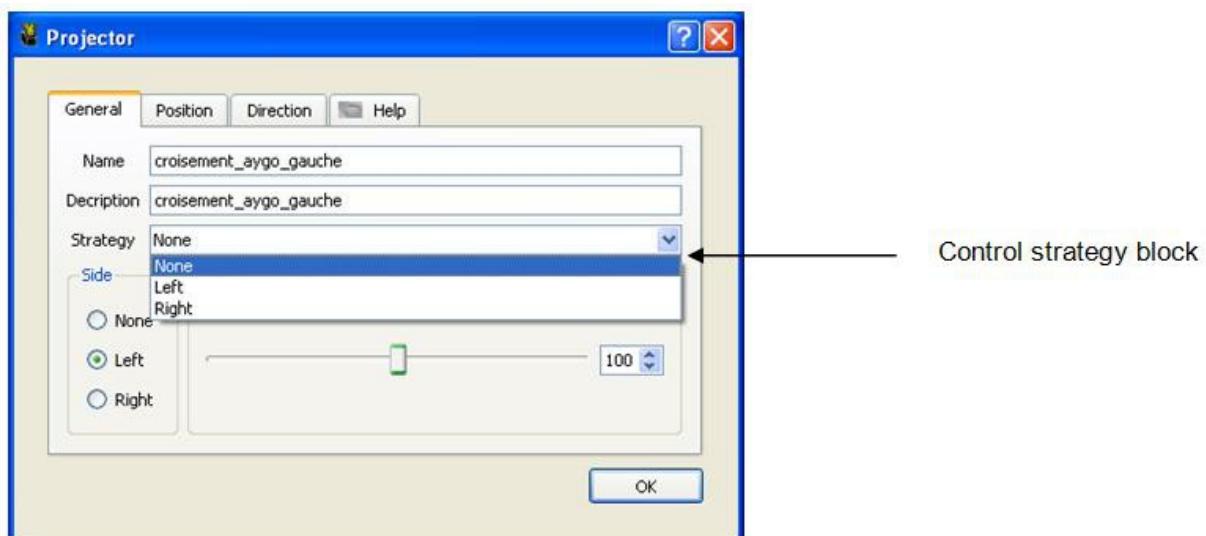


Illustration 114 : NightTestManager Software - Strategy

This action is optional; the control strategy block can also be assigned to the light sources contained in the projector.



If “None” is chosen, then the angles between the projector and its frame are null.

3. Select a light source, and choose a control strategy block

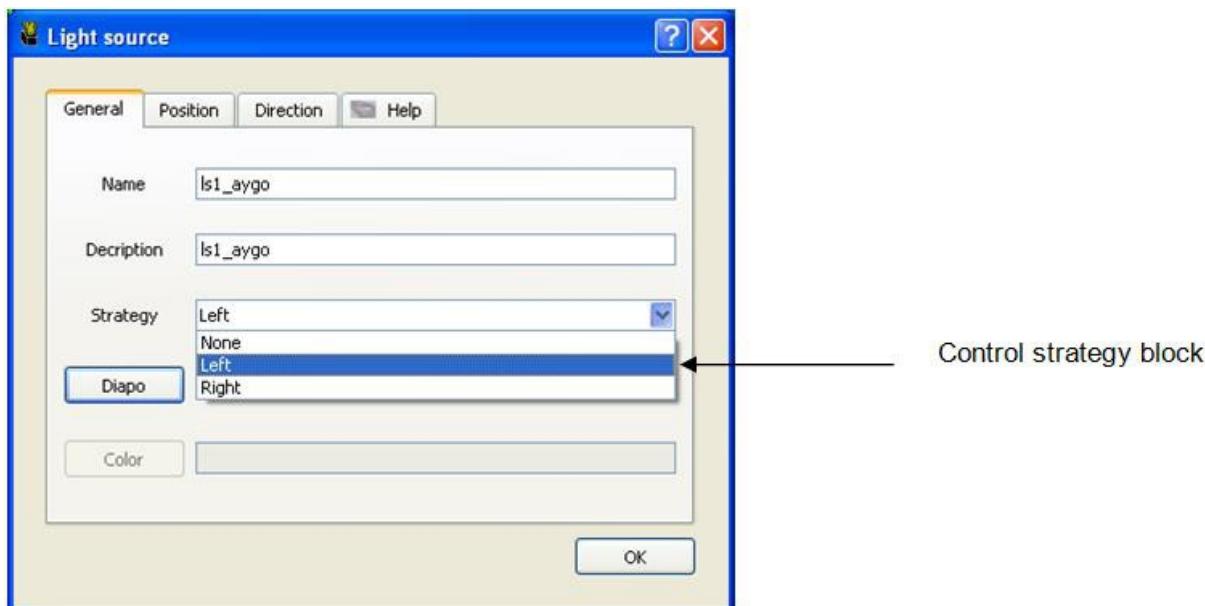


Illustration 115 : NightTestManager Software – Strategy - select

This action is optional; the control strategy block can also be assigned to the projector.



If “None” is chosen, then the angles between the light source and its frame, ie the project are null.



If the projector has a control strategy and not the light source: the intensity of the light source is the one of the projector.



If the projector has no control strategy and the light source has one: the intensity of the light source is the one of the light source.



If both the projector and the light source have a control strategy: the intensity of the light source is the one of the light source.

XIV.D.6.iv.c - Running a simulation

- Launch the SCANeR™ studio simulation as it is explained in the user manual. The scenario must include the AFS_inc.inc file.
- To launch/stop the AFS strategies, click on the run/stop button in the NightTestManager software (in the AFS Menu).

xv - VISUAL PLUGIN API

xv.a - INTRODUCTION

SCANeR™ studio Visual plugin API offers the ability to modify or increase the 3D representation of simulation.

Efforts have been made to allow C++ developers to gain access to SCANeR™ visual module and simulation data (weather conditions, vehicles properties, terrain, etc).

xv.a.1 - REQUIREMENTS

- Refer to the "Environment" chapter (page 18) and
- The visual module of SCANeR™ is based on the 3D graphics toolkit called OpenSceneGraph.



SCANeR™ studio embeds its own installation of OpenSceneGraph ([OSG](#)). This OpenSceneGraph version is: [3.0.1](#) (here, learn more about OSG: <http://www.openscenegraph.org/>).



The 3D formats compatible with SCANeR™ studio are all [native OpenSceneGraph file formats](#) (for example: OSGB, OSGX, OSGT, OSG, IVE).



OpenSceneGraph can also read (and write) 3rd party file formats (for example: DAE, 3DS, OBJ, FLT, DXF). SCANeR™ studio just uses plugins delivered with OpenSceneGraph to load or save such 3D file formats (here, learn more about 3D file formats supported and limitations: <http://www.openscenegraph.org/projects/osg/wiki/Support/UserGuides/Plugins>). Some 3D files that you can retrieve on internet, may not be compliant with the "Advanced" [RenderingMode](#) of the [VISUAL](#) module. The rendering mode may display the 3D file as white object. OKTAL is not liable to this issue. It's an OpenSceneGraph extra format loader limitation. To know more about:

1. the [RenderingMode](#), please refer to the Observer.cfg keywords described in the SCANeRstudio_SIMULATION_UserManual documentation,
2. the rendering issue refer to the [relevant forum topic](#).
3. the way to use a Sketchup plugin developed to export OSG file format compliant with all SCANeR™ studio rendering modes (also refer to the [relevant forum topic](#)).

Developments had to be compliant with this version of OpenSceneGraph.

OpenSceneGraph compiled with compiler described in the "Environment" chapter.

Ask OKTAL support if you need a pre-compiled version of OpenSceneGraph.

XV.A.2 - PLUG-IN ARCHITECTURE OVERVIEW

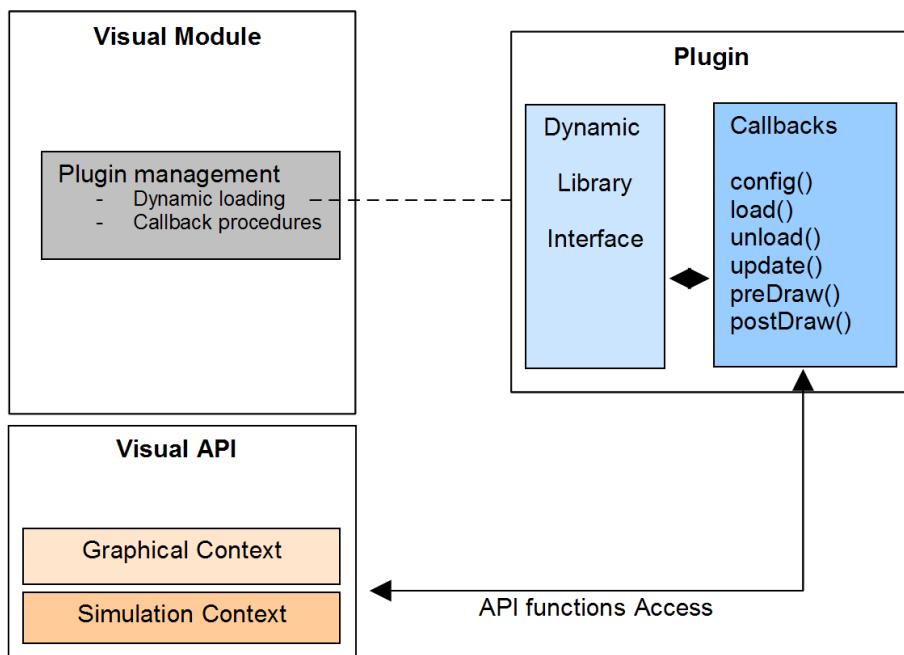


Illustration 116 : Plugin architecture

Plugins are collections of dynamic libraries composed of two parts:

- **interface** part will allow visual plug-in management to dynamically load each plugin
- **implementation** part composed of a set of call-back functions that will be called during each step of the rendering process:
 - **config**: is used to make user-defined initializations and configurations.
 - **load**: initialization is done and visual API can be accessed
 - **unload**: unloading of the plugin

At each SCANeR™ studio's step:

- **update** : called at each frame in the application process
- OSG frame for each channel:
 - OSG update: no action possible.
 - OSG culling: no action possible.
- **preDraw**: called before each frame with a valid OpenGL context
 - OSG Draw: no action possible.
- **postDraw**: called after each frame with a valid OpenGL context (see, for example, OsgGLPlugin
 - To draw objects as an overlay)

To access simulation kernel, plug-ins make use of the visual API functions.

Graphical Context class is intended to provide low level access to OpenSceneGraph graphical context, while Simulation data such as vehicle properties, weather conditions, database properties and many other tools can be accessed through the **Simulation Context** class.

XV.B - PLUGIN DEVELOPMENT

This section provides a quick tutorial on how to implement a basic plug-in.

For further information and plug-in examples developers may have a look to the API plug-in samples.

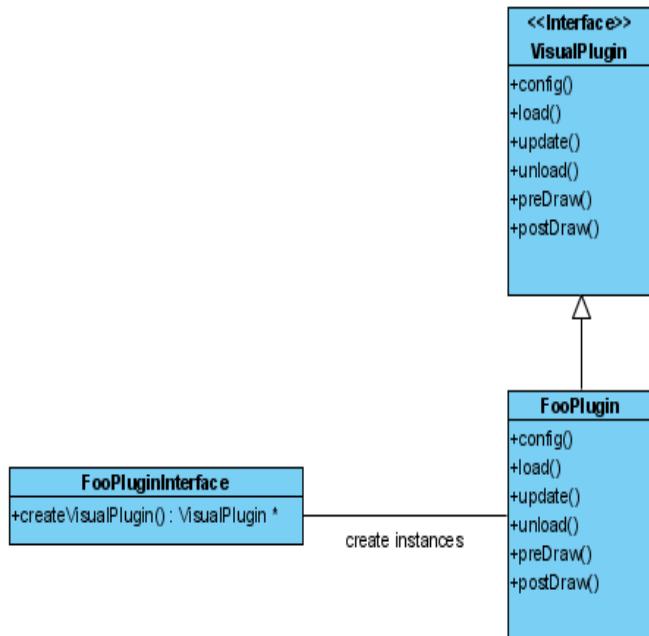


Illustration 117 : Plugin Class diagram

XV.B.1 - PLUG-IN INTERFACE

As far as plug-in are dynamic libraries, a plug-in interface is used to perform the loading process. Dynamic library must export the following function prototype:

- `VisualPlugin * createVisualPlugin();`

Function should return a pointer to an instance of your VisualPlugin class implementation.

XV.B.2 - PLUG-IN IMPLEMENTATION

As shown in Illustration 117 : Plugin Class diagram, a plug-in is implemented by deriving the VisualPlugin base class.

VisualPlugin class contains pure virtual functions that must be overridden.

Here is the body of a FooPlugin implementation class.

```
***** fooPlugin.h *****/
#include "VisualPlugin.h"

class GraphicalContext;
class SimulationContext;
class AnimatedBillboard;

class FooPlugin : public VisualPlugin
{
public:
    FooPlugin();
    virtual ~FooPlugin () {}
    virtual void config();
    virtual void load(GraphicalContext * p_visualContext, SimulationContext * p_simulationContext);
    virtual void unload();
    virtual void update();
    virtual void preDraw(VisualPlugin_Chanel* chan);
    virtual void postDraw(VisualPlugin_Chanel* chan);
};

***** fooPlugin.cpp *****/
#include "fooPlugin.h"

FooPlugin:: FooPlugin ()
{
    m_name = "fooPlugin";
    m_version = "V1.0";
    m_description = "does nothing";
    // TODO : add your code here
}
void FooPlugin::config()
{
    // TODO : add your code here
}
void FooPlugin::load(GraphicalContext * p_visualContext, SimulationContext * p_simulationContext)
{
    // TODO : add your code here
}
void FooPlugin::unload()
{
    // TODO : add your code here
}
void FooPlugin::update()
{
    // TODO : add your code here
}
void FooPlugin::preDraw(VisualPlugin_Chanel* chan)
{
    // TODO : add your code here
}
void FooPlugin::postDraw(VisualPlugin_Chanel* chan)
{
    // TODO : add your code here
}
```



In addition to `FooPlugin.cpp` and `FooPlugin.h`, you need an interface implementation too: `FooPluginInterface.cpp` and `FooPluginInterface.h`. Inspire yourself by modifying the project `<STUDIO_PATH>\<STUDIO_VERS>\APIs\VisualPluginAPI\samples\OsgPlugin`.

XV.B.3 - ACCESSING VISUAL REPRESENTATION USING API FUNCTIONS

Visual API is composed of two main objects: GraphicalContext and SimulationContext

These instance can be accessed in the visualPlugin::load(GraphicalContext * gc, SimulationContext *sm) call back function. We suggest you to keep a pointer on these objects for further use during the simulation.

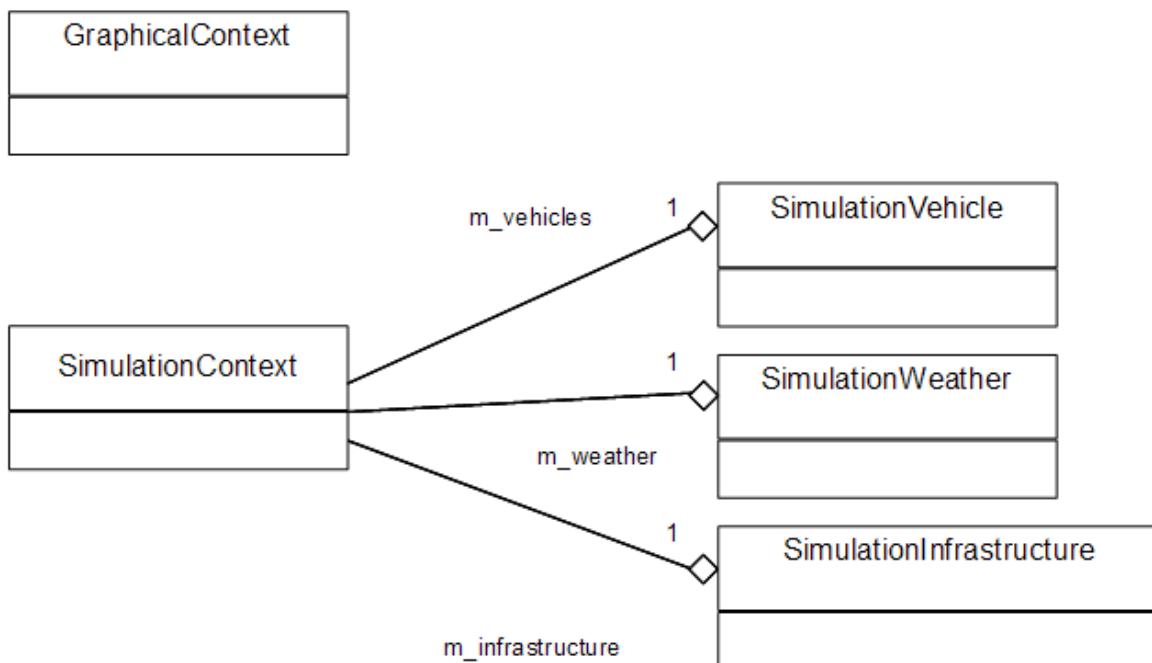


Illustration 118 : visual API class diagram

XV.B.3.i - Graphical context

The class provides low level access to the scene graph graphical context.

Context allows users to handle low level graphical components:

- Visual pipes and channels
- Visual scene
- Main Terrain Node

OKTAL does not advise to attach the objects directly to the root of the sceneGraph because intermediate nodes (specific to SCANeR™) can prevent seeing them. Add 3D objects under the node "3D Scene" and the images under the node "Scene 2D". An example shows how.

```
***** visualPluginUtils.h *****
#include <osg/Group>
```

```
***** visualPluginUtils.cpp *****
#include <string>
#include <osg/Group>
```

```

=====\brief Searches for the 3D scene node in the scene.
=====param scene The scene.
=====return null if it fails, else the found 3D scene.
=====osg::Group* findScene3D(osg::Group* scene);

=====\brief Searches for the 2D scene node in the scene.
=====param scene The scene.
=====return null if it fails, else the found 2D scene.
=====osg::Group* findScene2D(osg::Group* scene);

//return the scene 3D or NULL
osg::Group* findScene3D(osg::Group* scene)
{
//this is a traverse of the light sub-graph to find the group Scene 3D
if(scene == NULL)
    return NULL;

std::string nodeName = scene->getName();
if(nodeName.compare("Scene 3D") == 0)
{
    return scene;
}

for(unsigned int i=0; i< scene->getNumChildren(); i++)
{
    osg::Group* g = findScene3D(dynamic_cast<osg::Group*>(scene->getChild(i)));
    if(g)
    {
        return g;
    }
}
return NULL;
}

//return the scene 2D or NULL
osg::Group* findScene2D(osg::Group* scene)
{
//this is a traverse of the light sub-graph to find the group Scene 2D
if(scene == NULL)
    return NULL;

std::string nodeName = scene->getName();
if(nodeName.compare("Scene 2D") == 0)
{
    return scene;
}

for(unsigned int i=0; i< scene->getNumChildren(); i++)
{
    osg::Group* g = findScene2D(dynamic_cast<osg::Group*>(scene->getChild(i)));
    if(g)
    {
        return g;
    }
}
return NULL;
}

```

xv.B.3.ii - Simulation context

Simulation context provides access to simulation information:

- Weather (snow and rain intensity, cloud density...)
- Vehicles properties (geometry, position, speed...)
- Database,
- Infrastructures,

- Scenario data (Channel values: read and write access)

Specific methods are grouped in the subclasses **SimulationInfrastructure** / **Simulation Weather** / **SimulationVehicle** (see class reference for further details).

XV.B.4 - PLUG-IN EXAMPLES

Samples plug-in are provided to show the capabilities of the plug-in API and give some ideas on what developers can do with it.

Samples are located in the samples directory and include:

- **OsgGLPlugin**: Pre and post draw example using OpenGL
- **OsgInfrastructurePlugin**: Query informations from SimulationContext

XV.C - INSTALLATION NOTES

XV.C.1 - COMPILATION UNDER WINDOWS VISTA/7

Add the « APIs/VisualPluginAPI/include » folder as include search path in your Visual C++ project.

Add the « APIs/VisualPluginAPI/lib » folder as library search path in your Visual C++ project and link with the visualPluginApiLib.lib

XV.C.2 - PLUG-IN INSTALLATION:

Plug-ins dynamic libraries must be placed in the « plugins » subfolder of the SCANeR studio binaries “win32” folder.

Activation is managed simply by editing the « visualPlugin.cfg » configuration file of the current configuration.

```
; visualPlugin.cfg
; This configuration file defines a list of visual plugins
; !!! Remember plugin are rendered in the order they are placed in this file
;
VisualPlugin = OsgGLPlugin.dll OsgInfrastructurePlugin.dll
```

Table 39 : Example of visualPlugin.cfg file

XV.D - API FUNCTIONS REFERENCE

XV.D.1 - GRAPHICAL CONTEXT METHODS

int getNumPipeWindow()	get the number of active window
int getPWNumChannel(int p_pipeWinId)	get the number of channels for a given window
VisualPlugin_Chanel* getPWChannel(int p_pipeWinId, int p_chan)	Get a particular channel from a given window Id and channel Id. Note: Pipes and channels start at 1 into the observer.cfg. So parameters p_pipeWinId and p_chan are the id into the observer.cfg -1. Ex: Get the Pipe '[VISUAL:1]' and the channel [VISUAL:1.1] → getPWChannel(0, 0)
VisualPlugin_Scene* getScene()	Get the visual scene
VisualPlugin_Node* getTerrainNode()	Get the database terrain node
int getObserverCount() const	Get the total number of observers in the scene
VisualPlugin_Chanel* getObserverChannel(int n)	Get the corresponding channel of a specific observer

XV.D.2 - SIMULATION CONTEXT METHODS

SimulationVehicles * vehicles()	Get a pointer on the SimulationVehicles class
SimulationInfrastructure * infrastructure()	Get a pointer on the SimulationInfrastructure class
SimulationWeather * weather()	Get a pointer on the SimulationWeather class
const char* getTerrainName() const	Get the terrain name
void setExportChan(int p_chan, float p_val)	Send an exportChannel message to the SCANeR©II message queue.
float getExportChanValue(int p_chan)	get the current value on a particular channel of the SCANeR©II message queue.
void printNotifyMessage(const char* s)	Print message on the ERROR module console (green color)
void printDebugMessage(const char* s)	Print message on the ERROR module console (white color)
void printWarningMessage(const char* s)	Print message on the ERROR module console (yellow color)
void printBeepMessage(const char* s)	Print message on the ERROR module console and launch a beep sound.
void printErrorMessage(const char* s)	Print message on the ERROR module console (red color)
const char* utilsGetPath(const char* fileScannerPath)	Get the path to a SCANeR File according the given SCANeR Location.
const char* utilsGetValue(const char* file, const char* section, const char* token)	Get the value of the token in the given configuration file.

XV.D.3 - SIMULATION INFRASTRUCTURE METHODS

int getNumberOfObjects()	Get total number of infrastructure objects.
bool getObjectIndex(int objectIndex, string& name)	Get the object name from a given object index. Returns false if index doesn't exists.
int getObjectIndex(const string &objectName)	Get object index from a given object name. Returns -1 if name doesn't exists.

bool getObjectPosition(int objectIndex, double& X, double& Y, double& Z)	Get object position from a given object index. Returns false if object doesn't exists. X, Y, Z are absolute scene coordinates, unit is meter.
bool getObjectRotation(int objectIndex, double& H, double& P, double& R)	Get object rotation from a given object index. Returns false if object doesn't exists. H,P,R are absolute scene coordinates, unit is radian .
VisualPlugin_Node* getObjectTopNode(int objectIndex)	Get a visual node from a given object index. Returns NULL if object doesn't exists.

XV.D.4 - SIMULATION VEHICLES METHODS

bool vehicleExists(int p_vehId)	Check whether a given vehicle exists or not. Return false if not present.
std::string getVehicleType(int p_vehId)	Get the vehicle type name (ie: Renault Clio, Audi_TT, pedestrian...)
void getVehicleCoordinates(int p_vehId, double * p_x, double * p_y, double * p_z, double * p_h, double * p_p, double * p_r)	Get the vehicle absolute coordinates p_x, p_y,p_z : position in meters. p_h, p_p, p_r : rotation in radian.
VisualPlugin_Node* getVehicleTopNode(int vehId)	Get the visual node of the specified vehicle. Returns NULL if object doesn't exists.
float getVehicleSpeed(int vehId)	Get the vehicle speed in m/sec.
float getVehicleTurnSpeed (int vehId)	Get the vehicle turn speed in rad/sec.
float getVehicleAcceleration (int vehId)	Get the vehicle acceleration in m/sec2.
float getVehicleWheelsAngle (int vehId)	Get the vehicle wheel angle in radian, positive to the left.
int getVehicleNumWheel(int vehId)	Get the number of wheels of the specified vehicle.
int getFirstVehicleId()	Get the id of the first vehicle into your scenario (not necessary id = 0). Additional with getVehicleId(int p_vehId).
int getNextVehicleId(int p_vehId)	Get the id of the next vehicle after the p_vehId into your scenario.
float getVehicleLength(int p_vehId)	Get the length (from mdl file) of the specified vehicle. Return value is a distance in millimeters.
float getVehicleRearAxleDist(int p_vehId)	Get the distance between the center of rear wheel and car bottom (from mdl file) of the specified vehicle. Return value is a distance in millimeters.
void getVehicleEye(int p_vehId, float * p_x, float * p_y, float * p_z)	
float getVehicleDistToFront(int p_vehId)	Get the distance between center of rear wheel and car front (from mdl file) of the specified vehicle. Return value is a distance in millimeters.
string getSensorConfigurationName(int p_vehId) const	Get the name of the sensor configuration attached to the specified vehicle.

XV.D.5 - SIMULATIONWEATHER

float getSnowIntensity()	Get current snow intensity
--------------------------	----------------------------

	Return value is between [0..1]
float getRainIntensity()	Get current rain intensity Return value is between [0..1]
float getWaterOnRoad()	Get the water accumulation on the road. Return value is between [0..1]
float getWindHeading()	Get current wind heading Unit is radian.
float getWindSpeed()	Get current snow Intensity Unit is km/h.
float getFogDistance()	Get current fog distance Return value is a distance in meters.
float getSkySaturation()	Get current sky saturation Return value is between [0..1]
float getSkyCloudsDensity()	Get current clouds density Return value is between [0..1]
double getDayTime()	Get current time of day. Return value is between [0..24] – unit is hour
void setSkyColor(float r, float g, float b)	Set the sky color value.

xvi - IMAGE SHARING API

xvi.a - API PRESENTATION

In general terms the image sharing API allows the user to share images between applications by basically creating a pool of images. These can then be updated at a given frequency thus generating an image stream.

A particular purpose of the API is to allow a user to retrieve images rendered by the SCANeR Visual and CameraSensor processes. This way, a user module can receive at a certain rate the rendered images and do some image processing frame per frame.

It is also possible for the user to share his own images and, via a SCANeR visual plugin, retrieve and display them in the visual process.

xvi.b - How DOES IT WORK

The API works similarly to the communication controller in the SCANeR API (if we see the image pool as an equivalent of the network/shm):

- **Initialization:** The user can declare several input images (coming from another application) and output images (own images to be shared) and give them a name that will identify them.
- **Update images to read:** Updates the data of input images from the image pool.
- **Get/Set data:** Get the raw data and description from input images and setting the data for output images using the images identifiers.
- **Update images to write:** update the data of output images so they can be then retrieved by other applications.

An image is composed of a header holding information about the image such as width, height, number of components per pixel (Default RGB), etc. and the image raw data.

Two modes can be used for retrieving or “reading” images:

xvi.b.1 - Non-blocking reading mode

The user updates the input images at a time T. Every image is retrieved as is at that moment. The update frequency depends on the caller application. Depending on this frequency some images might not change. The update is done when all images are retrieved or a time-out goes to 0.

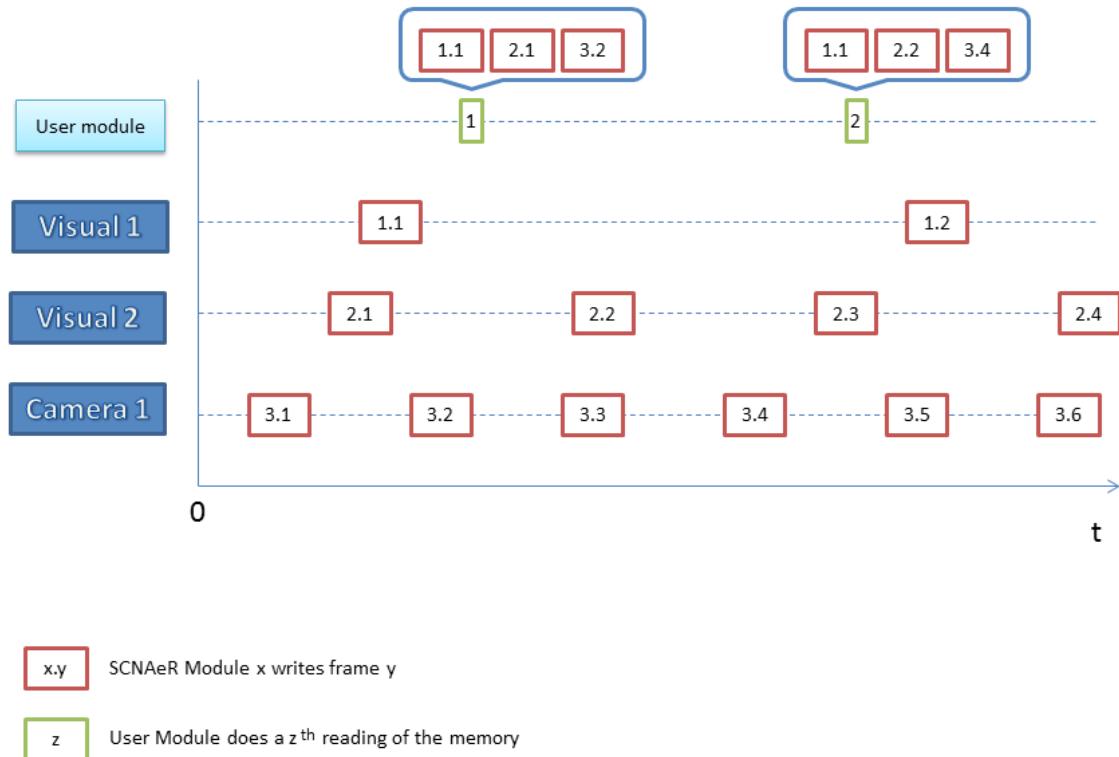


Illustration 119: Image Sharing API Non-blocking reading

xvi.B.2 - Blocking reading mode

The user updates the input images at a time T. The application waits until every image has a new frame (this blocks the writing applications until every writing application has written a new frame) or a time-out goes to 0. This is used to ensure that at every update all images have new data. This is a way of synchronising the image sharing.

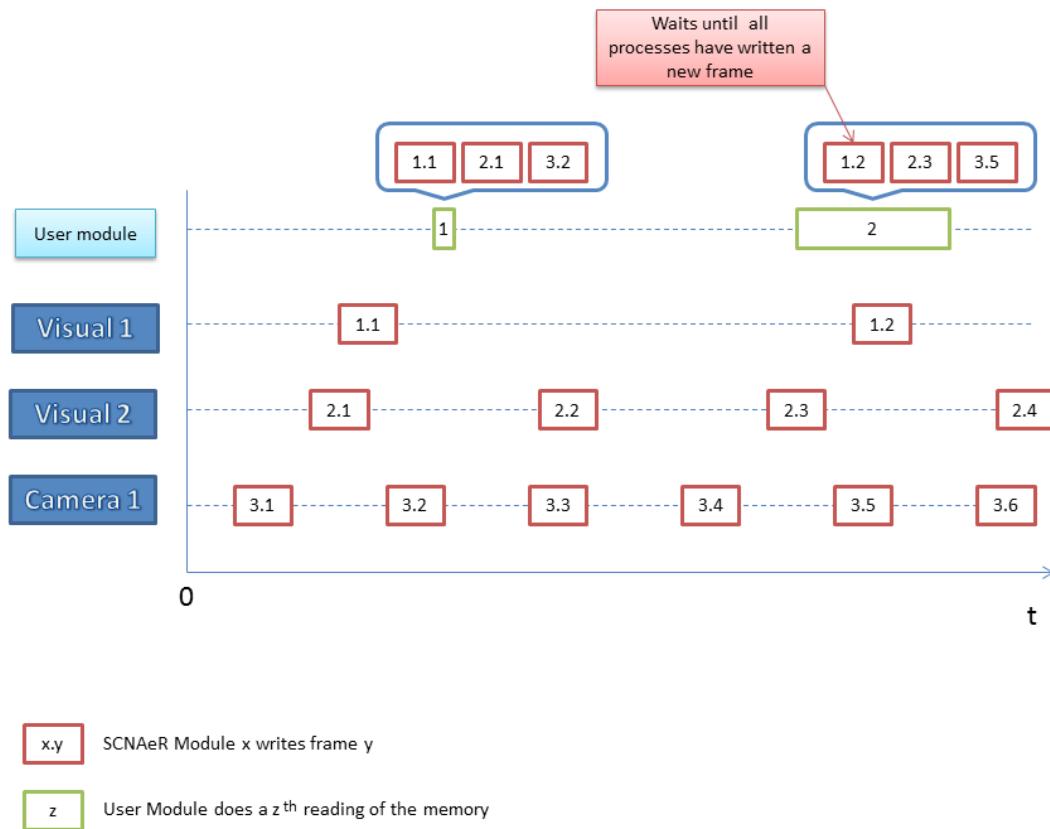


Illustration 120: Image Sharing API Blocking reading

xvi.c - API DESCRIPTION

xvi.c.1 - ENUMERATIONS

ImgShError
NO_IMG_ERROR
COULD_NOT_READ_WRITE
IMAGE_NOT_DECLARED
NO_IMAGES_DECLARED

ImgShFormat
IMG_SH_RGB
IMG_SH_RGBA
IMG_SH_BGR

IMG_SH_BGRA

xvi.c.2 - Structures

ImgShHeader
<code>int myFrameNumber</code>
<code>double myTimestamp</code>
<code>ImgShFormat myImageFormat</code>
<code>long myImageWidth</code>
<code>long myImageHeight</code>
<code>long myImageNumComp</code> (Number of components RGB = 3, RGBA = 4)

ImgShData
<code>ImgShHeader myHeader</code>
<code>char* myImgRawData</code>

xvi.c.3 - Functions

Function name	Description
<code>ImgShData* ImgSh_declareInputImage (const char * imageName)</code>	Brief: declares an image to be read Param 1: imageName image. It serves as the image identifier Return: true if the image was created, false otherwise
<code>ImgShData* ImgSh_declareOutputImage (const char * imageName, int width, int height,</code>	Brief: declares an image that is going to be written and defines the image properties from the given header Param: imageName Name of the iamge Param: width image width Param: height image height

<pre>ImgShFormat imageFormat = IMG_SH_RGB, int maxImageSize = 1920/*width*/*1080/*height*/*4/*components per pixel*/)</pre>	<p>Param: imageFormat image format (RGB, BGR, RGBA, BGRA)</p> <p>Param: maxImageSize limits the total size an image can have. Default is 8294400</p> <p>Return: true if the image was created, false otherwise</p>
<pre>ImgShError ImgSh_updateOutputImages()</pre>	<p>Brief: Writes all the image data</p> <p>Return: error if writing failed</p>
<pre>ImgShError ImgSh_updateInputImages(int waitForNewImages = 0, int timeOut = -1/*infinite*/)</pre>	<p>Brief: Updates the input images. This function can block trying to retrieve the images. A timeout can be specified to avoid long blockings</p> <p>Param: waitForNewImages waitForNewImages If positive the function must wait until all the requested images have increase their image number or the timeout (if any) runs out.</p> <p>If 0 or negative, the function retrieves the last image whether its number has changed or not. It stops when all the images have been accessed or the timeout (if any) has run out.</p> <p>Param: timeOut Reading timeout</p> <p>Return: error if an image could not be accessed for reading</p>
<pre>ImgShData* ImgSh_getImageData (const char* name)</pre>	<p>Brief: Returns the given image data</p> <p>Param: name image name</p> <p>Return: pointer to the data structure</p>
<pre>ImgShError ImgSh_deleteImage (const char* name)</pre>	<p>Brief: Deletes the image identified by the given name</p> <p>Param: name image name</p> <p>Return: error if the output image was not declared or not image at all is declared</p>
<pre>ImgShError ImgSh_deleteAllImages()</pre>	<p>Brief: Deletes all the declared images</p> <p>Return: error if no image at all was declared</p>
<pre>const char * ImgSh_getLastErrorMessage()</pre>	<p>Brief: Retrieves last error string</p> <p>Return: last error string</p>
<pre>int ImgSh_getMaxImageSize (const char* name)</pre>	<p>Brief: Retrieves an image maximum size</p> <p>Param: name image name</p> <p>Return: image max size</p>

XVI.D - STREAMING FROM VISUAL AND CAMERA SENSOR

It is possible to enable image sharing from the SCANeR Visual and Camera Sensor processes:

XVI.D.1 - VISUAL

In the observer.cfg configuration file (see the Image rendering section in the SCANeR_SIMULATION_UserManual for more information) the keyword “FramesSharingFormat” can be added. By default its value is set to “NoFrameSharing”.

This key word defines the format of the images that will be shared (RGB, RGBA, BGR, BGRA).

When declared, an image has a default maximum total size of 8294400 bytes (suitable for a 1920 x 1080 image in RGBA or BGRA format). This can be overridden by giving the desired value to the keyword “FrameShareMaxSize”.

XVI.D.2 - CAMERASENSOR

To activate image sharing for a camera sensor equipped in a vehicle:

Open the Sensor configuration editor by

- right-clicking the vehicle and selecting “Edit sensor configuration...”
- or opening the VehicleInstanceSetup dialog (double click the vehicle or right-click vehicle->“Edit instance...”) and in the Sensors tab, select the “Edit sensors configuration” (a sensors configuration must be selected).

See the sensors configuration section in the SCANeR_SCENARIO_UserManual for more information.

Select the camera sensor, check the image sharing option and define the image format:

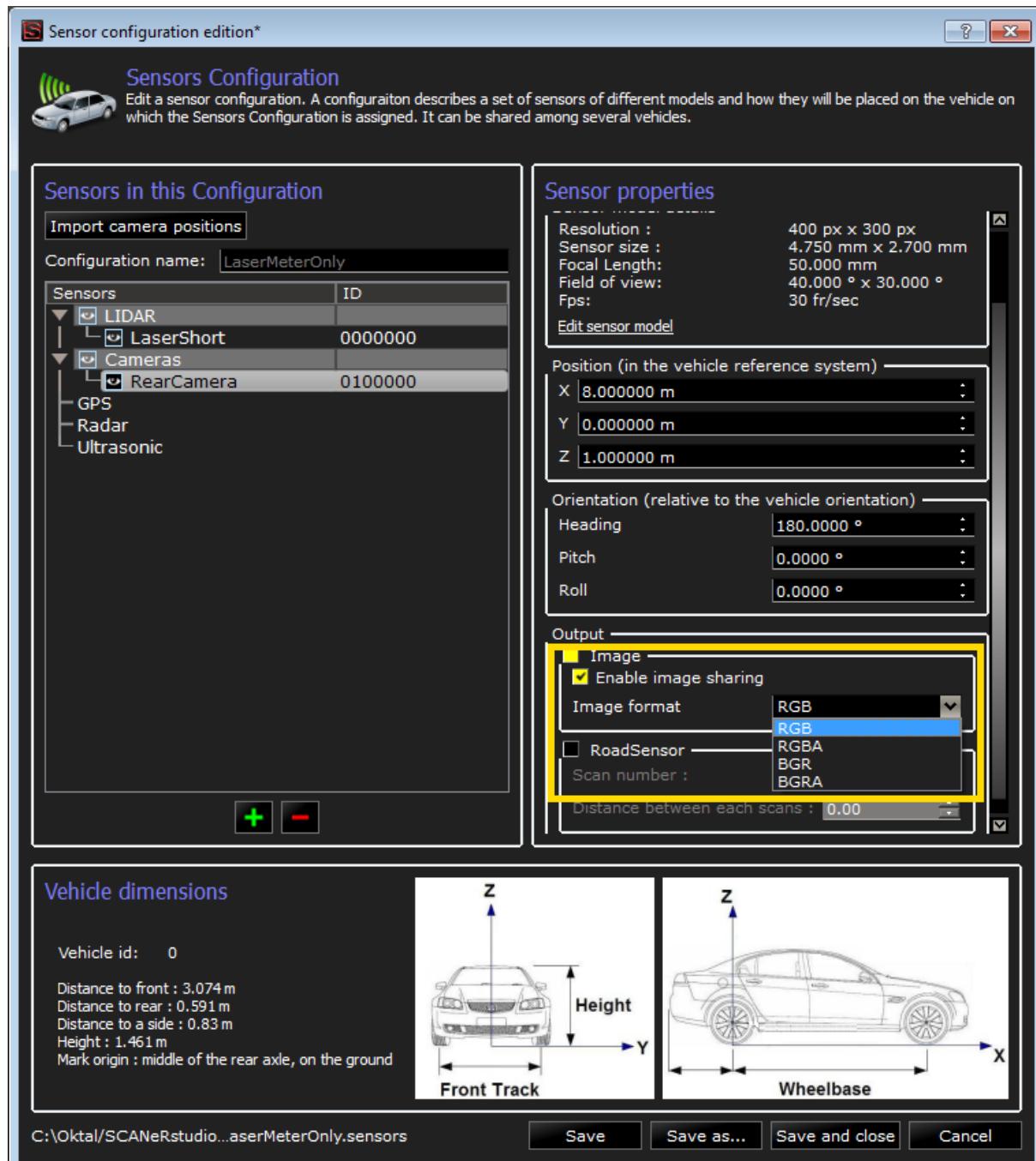


Illustration 121: ImageSharing Enabling sharing in a camera sensor

The maximum image size can be overridden in the cameraSensor.cfg configuration file the same way that is done in the observer.cfg file.

XVI.D.3 - CONVENTIONS

Once the Image Sharing is enabled, the user can receive the images from an application like an user module for example.

Visual images are identified with the names of the form:
 <PROCESS_NAME>:<PIPE_NUMBER>:<CHANNEL_NUMBER>

Camera sensor images are identified with the names of the form:

<PROCESS_NAME>:<SENSOR_GLOBAL_ID>

(Use the SENSOR_GLOBAL_ID as displayed in the sensors configuration editor dialog)

XVI.D.4 - EXAMPLE

A basic SCANeR™ Studio module receiving an image stream from a camera sensor with global id "0100000" would look like this "main()" function:

```
int main(int argc,char* argv[])
{
    /*! Initialization of the Process */
    Process_Init(argc, argv);
    /*! Process State */
    APIProcessState status;

    ImgShData* srcImg = NULL;

    do
    {
        /*! Wait for synchronization at the given frequency */
        Process_Wait();

        /*! Run the process */
        Process_Run();

        /*! Udate the status */
        status = Process_GetState();

        if (status == PS_LOAD)
        {
            srcImg = ImgSh_declareInputImage("CAMERASENSOR:0100000");
        }

        if (status == PS_RUNNING)
        {
            ImgShError error = ImgSh_updateInputImages();

            if (error == NO_IMG_ERROR )
            {
                // Retrieve the image header
                const ImgShHeader& srcImgHeader = srcImg->myHeader;

                // Add user code, image treatement on srcImg->myImgRawData
            }
        }
    } while (status != PS_DEAD);

    ImgSh_deleteAllImages();

    Process_Close();
}
```

XVI.D.5 - IS_VIDEOCAPTURE SAMPLE DESCRIPTON

The IS_VideoCapture sample is a SCANeR™ Studio module. It capture a webcam video stream or a video file and copy it as an Image Sharing API image.

This module can be used with the VideoMapping visual plugin to create live video texture in the visual scene.

The configuration is done in the “imageSharingVideoCapture.cfg” file.

imageSharingVideoCapture.cfg

```
; DEBUG OPTIONS
showCapturedImage = false
showSHMImage = false
; INPUT PARAMETERS
captureType = CAPTURE_FROM_CAM
;captureType = CAPTURE_FROM_FILE
cameraIndex = 0
videoFileName = path/to/video/file
; PARAMETERS USED ONLY WITH CAMERA OR ACQUISITION CARD
; THEY WILL BE IGNORED WITH VIDEO FILES
captureFrameWidth = 800
captureFrameHeight = 600
captureFrameRate = 30
; OUTPUT PARAMETERS
shmImageName = VirtualHUDSharedImage
shmImageWidth = 800
shmImageHeight = 600
flipHorizontally = false
```

Keyword	Type	Meaning
showCapturedImage	Boolean	Show the captured image in a debug window
showSHMImage	Boolean	Show the image in the Image Sharing API in a debug window
captureType	Enum	2 Values possible: CAPTURE_FROM_CAM: capture image from a webcam.

		CAPTURE_FROM_FILE: capture the image to a video or an image file.
cameraIndex	Intenger	Only used with captureType = CAPTURE_FROM_CAM, Index of the camera, started at 0.
captureFrameWidth	Intenger	Only used with captureType = CAPTURE_FROM_CAM, Width of the webcam image.
captureFrameHeight		Only used with captureType = CAPTURE_FROM_CAM, Height of the webcam image.
captureFrameRate = 30		Only used with captureType = CAPTURE_FROM_CAM, Frame rate of the webcam to use.
videoFileName	String	Only used with captureType = CAPTURE_FROM_FILE, Path to the video or image file.
shmlImageName	String	Image name in the Image sharing API
shmlImageWidth	Integer	Image width in the Image sharing API
shmlImageHeight	Integer	Image height in the Image sharing API
flipHorizontally	Boolean	If true, filp horizontally the image.

XVI.E - LIMITATIONS AND CONSTRAINTS

- When using the API sender and receiving modules must be in the same machine
- Enabling Image Sharing for the visual and camera sensor processes can affect the performances according to the size of the rendered channels.

XVII - UDP RTGATEWAY PROTOCOL

XVII.A - INTRODUCTION

This protocol defines how to connect a custom made program to RT_GATEWAY module. This is an alternative to writing a SCANeR API module. The advantage of this solution is that the program will not have dependencies to SCANeR API, so it can be used on all SCANeRstudio version (from 1.6) without modification/recompiling. The UDP program can be used on different OS (Windows, MAC, Linux).

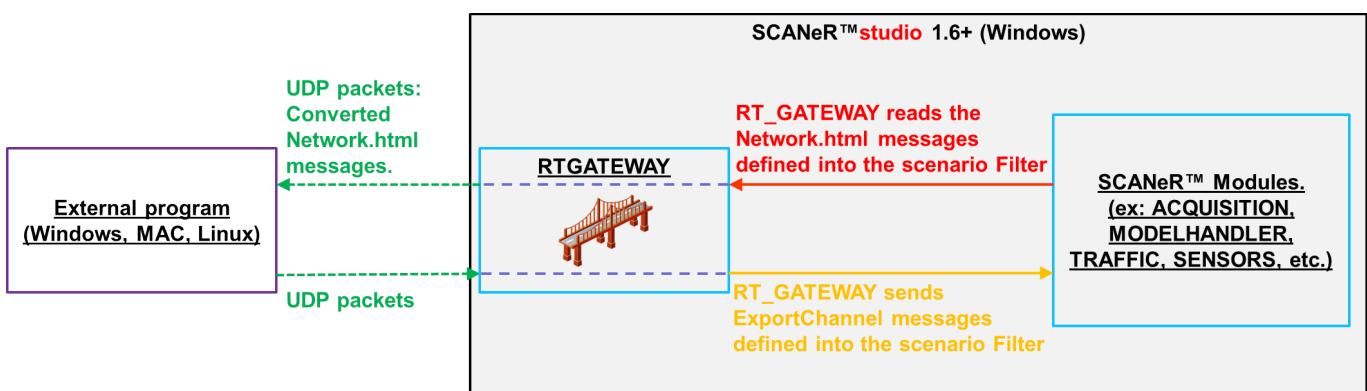


Illustration 122 : RT_GATEWAY protocol

XVII.B - PROTOCOL

XVII.B.1 - SOCKET INITIALISATION

The UDP program must open a listening UDP socket.

XVII.B.2 - RECEIVE PACKET FORMAT

The format of the UDP packets that the UDP program will receive is one array of N double (floating point values) elements.

To know how are sorted the selected elements into the UDP packet, the user must export the mapping to a CSV file format. To do so click on the “**Export mapping to CSV**” button.

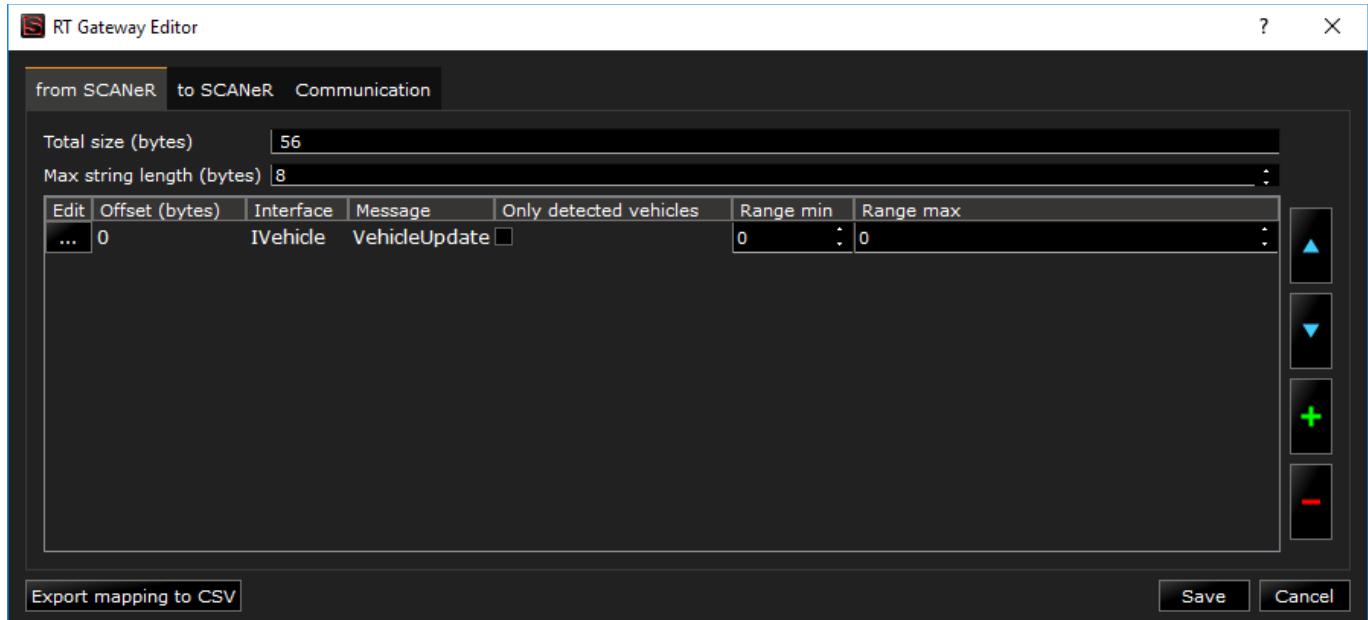


Illustration 123 : Network.html messages to be exported on UDP

Once exported the CSV file format is the description of the array elements:

	A	B
1	memory buffer offset(bytes)	name
2		0 speed[0]
3		8 speed[1]
4		16 speed[2]
5		24 speed[3]
6		32 speed[4]
7		40 speed[5]
8		48 engineSpeed

Illustration 124 : Data mapping on UDP packet

XVII.B.3 - SEND PACKET FORMAT

The “send” packet is the same format as the “receive” packet.

XVII.B.4 - EXPECTED BEHAVIOR

The UDP program waits on its socket for UDP packets from RT_GATEWAY and sends UDP packets to RT_GATEWAY module on that socket.

XVII.C - SCANeR SETUP

Once the UDP program is developed, it should be launched. Once the UDP program is running, set up the host name and port number in the filter used by RT_GATEWAY. Once this is done, the communication between the UDP program and RT_GATEWAY is enabled.

XVII.D - UDPRTGATEWAYIO SAMPLE

XVII.D.1 - ENVIRONMENT

The UDPRTGatewayIO has been developed in:

- C/C++ using Windows libraries, sources are available under “<STUDIO_VERSION>\SCANeRstudio_1.X\APIs\samples\UDPRTRGatewayIO”,
- Simulink, the model is available under “<STUDIO_VERSION>\SCANeRstudio_1.X\APIs\samples\UDPRTRGatewayIO\Simulink”.

XVII.D.2 - BEHAVIOUR

UDPRTRGatewayIO sample has been made to be used with the default scenario Studio_RTGateway.

It reads UDP packets send by the default Filter UDP.

It sends UDP packets on export channels 2000, 2001 and 2002.

By default, this socket use the port 8192. If needed the port number can be specified using the argument “-port”.

Note: With Simulink edit the block “Subsystem1\UDP Receive” and the field “Local IP port” to setup the port.

When it receives a packet from RT_GATEWAY the UDP program simply echoes the values on the standard output (the text console).

When the default scenario Studio_RTGateway receives data from the UDP program it displays Export channels values into the APPLICATION LOG.

XVII.D.3 - COMPILING

To compile the UDPCabin sample, open the “complete” sample solution in Visual studio and compile the UDPCabin sample.

Note: With Simulink you can build the model or run a Co-simulation.

XVII.D.4 - TESTING

Test the UDPRTGatewayIO sample:

Prerequisites:

RT_GATEWAY module is protected by a specific license so if you are interested in evaluation please send an e-mail to support-scaner@oktal.fr.

Open the scenario Studio_RTGateway and start the modules:
TRAFFIC, MODELHANDLER, SCENARIO and RT_GATEWAY.

In this scenario,

- 1 vehicle id 0, it is the main vehicle, it is a CALLAS vehicle, driven by TRAFFIC.
- 1 filter "UDP", it is configured to:
 - export SCANeR network messages

- IVehicle\VehicleUpdate\Speed[6]
- IVehicle\VehicleUpdate\engineSpeed
- receive UDP packets on Export Channels 2000, 2001 and 2002
- send/receive data on UDP: <localhost@8192>
- The scenario script displays the Export Channels values into SCANeR APPLICATION LOG.

XVII.D.5 - MODIFYING

Once the sample is working, to implement your own UDP program, copy the sample to a different folder, rename it and replace the signal generation code in the sample with your own code (*if you use another OS than Windows the dependencies must be updated*).

xviii - UDP ACQUISITION PROTOCOL

xviii.a - INTRODUCTION

This protocol defines how to connect a custom made Acquisition module to SCANeR studio's HumanDriver. This is an alternative to writing a SCANeR API acquisition. The advantage of this solution is that the standard mapping and calibration tools of the HumanDriver module can be used instead of being implemented in the acquisition. The other advantage of this solution is that the acquisition can be deployed once and work on all SCANeR versions without modification/recompiling.

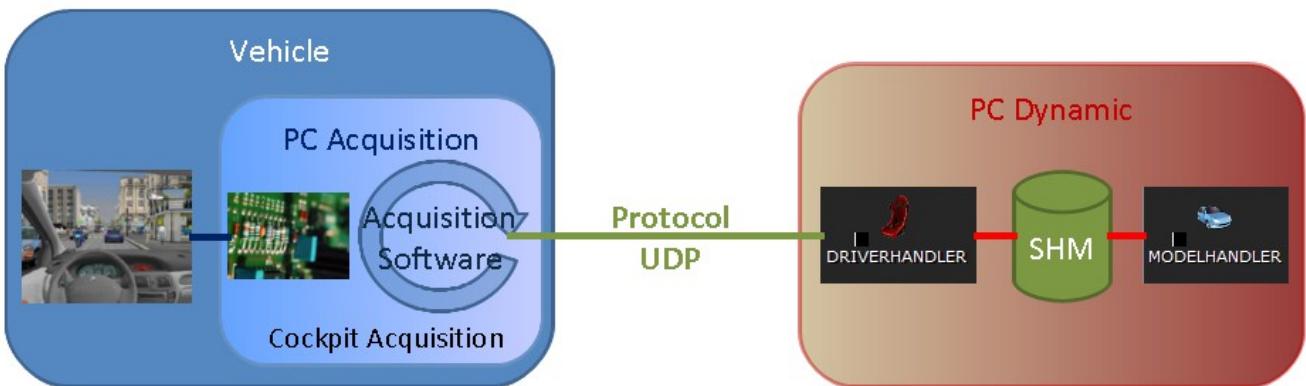


Illustration 125 : UDP Acquisition Architecture

xviii.b - PROTOCOL

xviii.b.1 - SOCKET INITIALISATION

The UDP acquisition module must open a listening UDP socket. By default, this socket should be 3232, but it should be possible to let the user choose a different socket.

xviii.b.2 - RECEIVE PACKET FORMAT

The format of the UDP packets that the UDP acquisition will receive is the following, in binary, little endian encoding:

one 32 bit int containing N, the number of floating point values being received, followed by N 32 bit float values, followed by a 32 bit integer containing M, the number of 32 bit integers received, followed by M 32 bit integer values.

For example, the UDP packet could contain:

the 32 bit little endian representation of 3 followed by three 32 bit floating point values followed by the 32 bit integer 0 (no integer values).

The size of the packet is sufficient to accommodate all the data but might be larger to contain additional, non documented data.

XVIII.B.3 - SEND PACKET FORMAT

The “send” packet is the same format as the “receive” packet.

XVIII.B.4 - EXPECTED BEHAVIOR

The UDP acquisition module waits on its socket and each time it receives a packet on that socket, it should process the data in the packet and respond with a data acquisition packet to be sent to the emitter of the received packet.

XVIII.C - SCANE R SETUP

Once the UDP acquisition module is developed, it should be launched on the host that contains the acquisition hardware. Once the UDP acquisition module is running, set up the host name and port number in the SCANE R studio HumanDriver module and enable UDP communication, as specified in the HumanDriver documentation. Once this is done, it should be possible to see the UDP acquisition host as a “Joystick” in the Joystick mapping dialogue.

XVIII.D - UDPCABIN SAMPLE

XVIII.D.1 - BEHAVIOUR

The UDPCabin does no real acquisition, it just sends floating values to the HumanDriver module. The values are, in order:

v
sin(v)
v x 2

where v is approximately (time in seconds/10).

When it receives a packet from HumanDriver, it simply echoes the values on the standard output (the text console).

XVIII.D.2 - COMPILING

To compile the UDPCabin sample, open the “complete” sample solution in Visual studio and compile the UDPCabin sample.

XVIII.D.3 - TESTING

To test the UDPCabin sample, launch it. Then, open the HumanDriverEditor tool available in the “Tools” menu in SCANE R studio. Edit a HumanDriver “*.hdrv” file or create a new one with “UDP protocol” enabled and specify the host and port the UDPDriver sample is running on. If the sample is running on the same PC as the HumanDriverEditor, just set “localhost” as host.

XVIII.D.4 - MODIFYING

Once the sample is working, to implement your own UDP acquisition, copy the sample to a different folder, rename it and replace the signal generation code in the sample with your own acquisition code.

XVIII.E - UDP_CABIN_ACQUISITION SAMPLE

XVIII.E.1 - DESCRIPTION

This sample is made with LabView 2011.

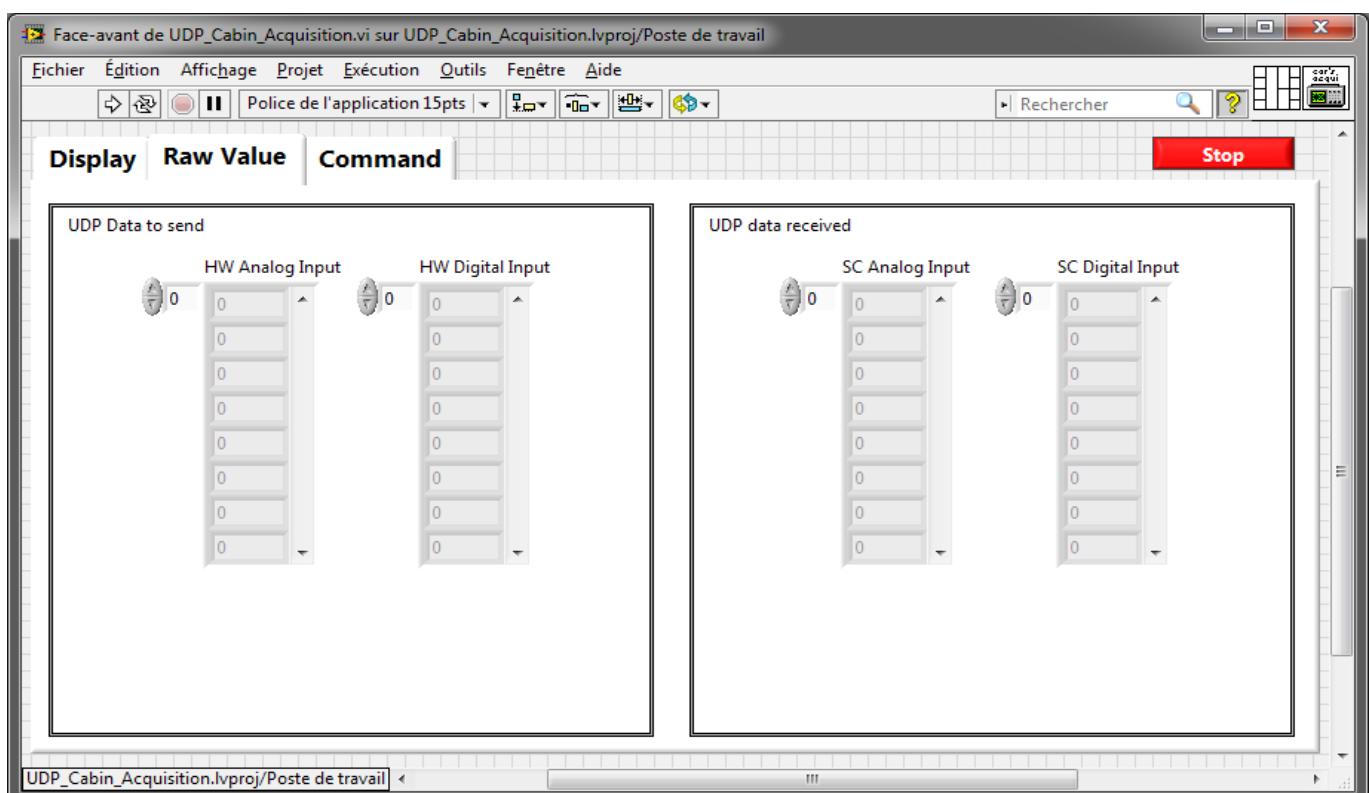


Illustration 126 : Cabin_Acquisition GUI

The Cabin_Acquisition sample makes the link between the acquisition card and the SCANeR™ studio HumanDriver module by the UDP protocol as follow :

- It measures the 8 Analog Inputs (from AI0 to AI7) and sent the data on the UDP
- It measures the 8 first Digital Inputs (from P0,0 to P0,7) and sent the data on the UDP
- It reads 2 float data from UDP and applies them to the Analog Output (AO0 and AO1)
- It reads 4 integer data from UDP and applies them to the 4 Digital Output (from P1,0 to P1,3)

Data send to UDP		Data receive from UDP	
Data	Description	Data	Description
8	Size of float data to send	2	Size of float data to send
8	Size of integer data to send	4	Size of integer data to send
AI0	Data from Analog Input 0	AO0	Data apply to Analog Output 0
...	Data from Analog Input 1, ..., 6	AO1	Data apply to Analog Output 0
AI7	Data from Analog Input 7	P1.0	Data apply to Digital Output P1.0
P0.0	Data from Digital Input 0	...	
...	Data from Digital Input 1, ..., 6	P1.3	Data apply to Digital Output P1.3
P0.7	Data from Digital Input 7		

Table 40 : Data Cabin_Acquisition exchange on UDP

XVIII.E.2 - HARDWARE

The sample uses NI USB 6008 acquisition cards (8 AI, 2 AO and 12 DIO) from National Instrument

To be recognized by the sample, this acquisition card has to have the name “Dev1” (for Device 1) in the Measurement & Automation Explorer (NI MAX)

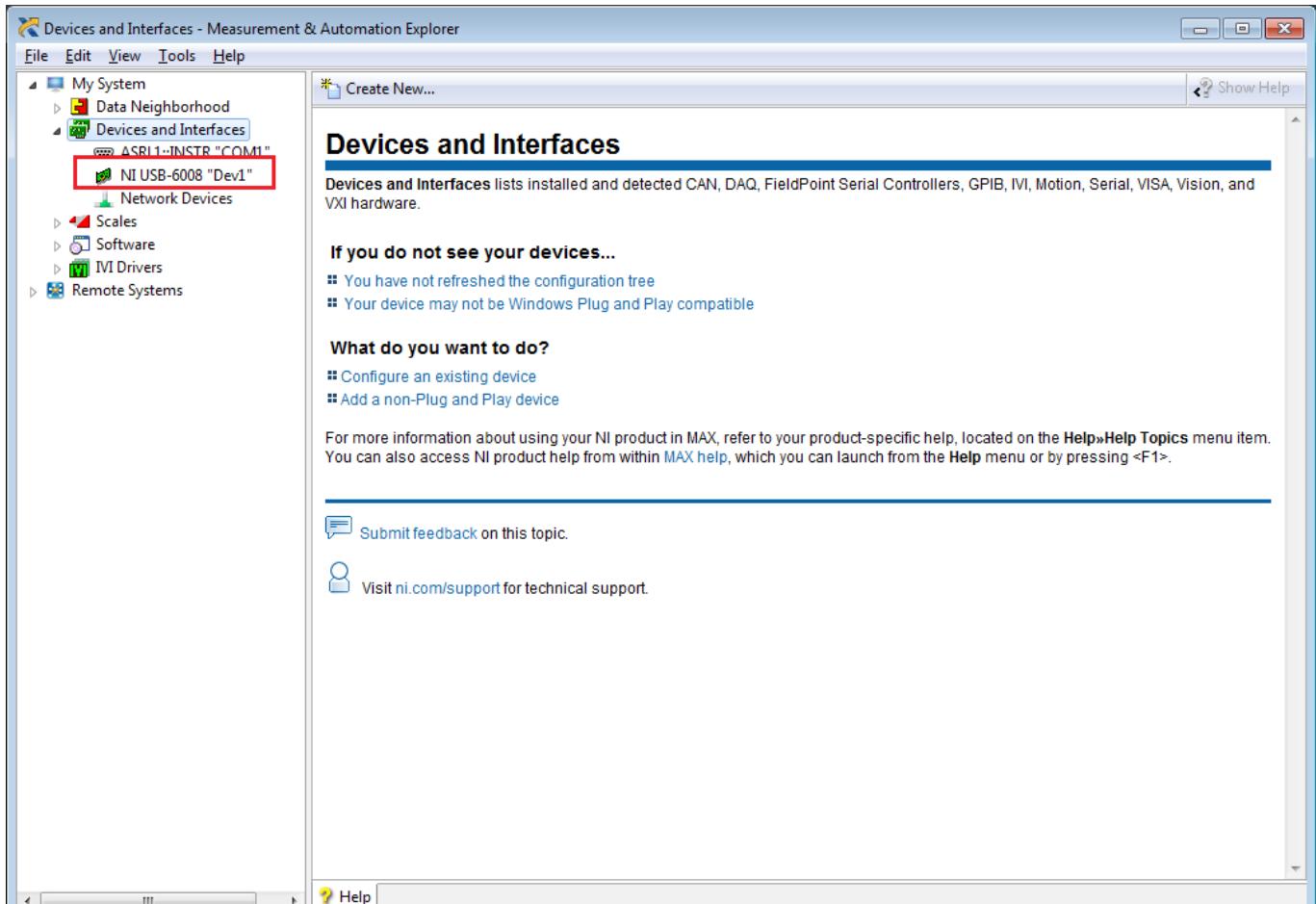


Illustration 127 : NI Measurement & Automation Explorer (NI MAX)

XVIII.E.3 - CONFIGURATION FILE

The UDP_Cabin_Acquisition.exe sample use a configuration file : Cabin_Acquisition.cfg

This file has to be placed in the same folder than the UDP_Cabin_Acquisition.exe software.

It contains the following information:

[Generality]	
Frequency = 10	<i>; is the software main loop frequency</i>
[UDP_Information]	
Port = 3232	<i>; is the UDP port number</i>

XVIII.E.4 - COMPILING

To compile the Cabin_Acquisition sample, open the UDP_Cabin_Acquisition.lvproj with LabView 2011 or later and compile the Cabin_Acquisition sample.

XVIII.E.5 - TESTING

To test the UDPCabin sample, launch the binary outside of the SCANeR™ studio context (a DOS console displays messages):

listening on port 3232...
sending packet....
...

Open the HumanDriverEditor tool available in the “Tools” menu in SCANeR studio. Edit a HumanDriver “*.hdrv” file or create a new one with “UDP protocol” enabled and specify the host and port (given in the DOS console) the UDPDriver sample is running on. If the sample is running on the same PC as the HumanDriverEditor, just set “localhost” as host.

Set data to send to the UDPCabin (and save the HDRV):

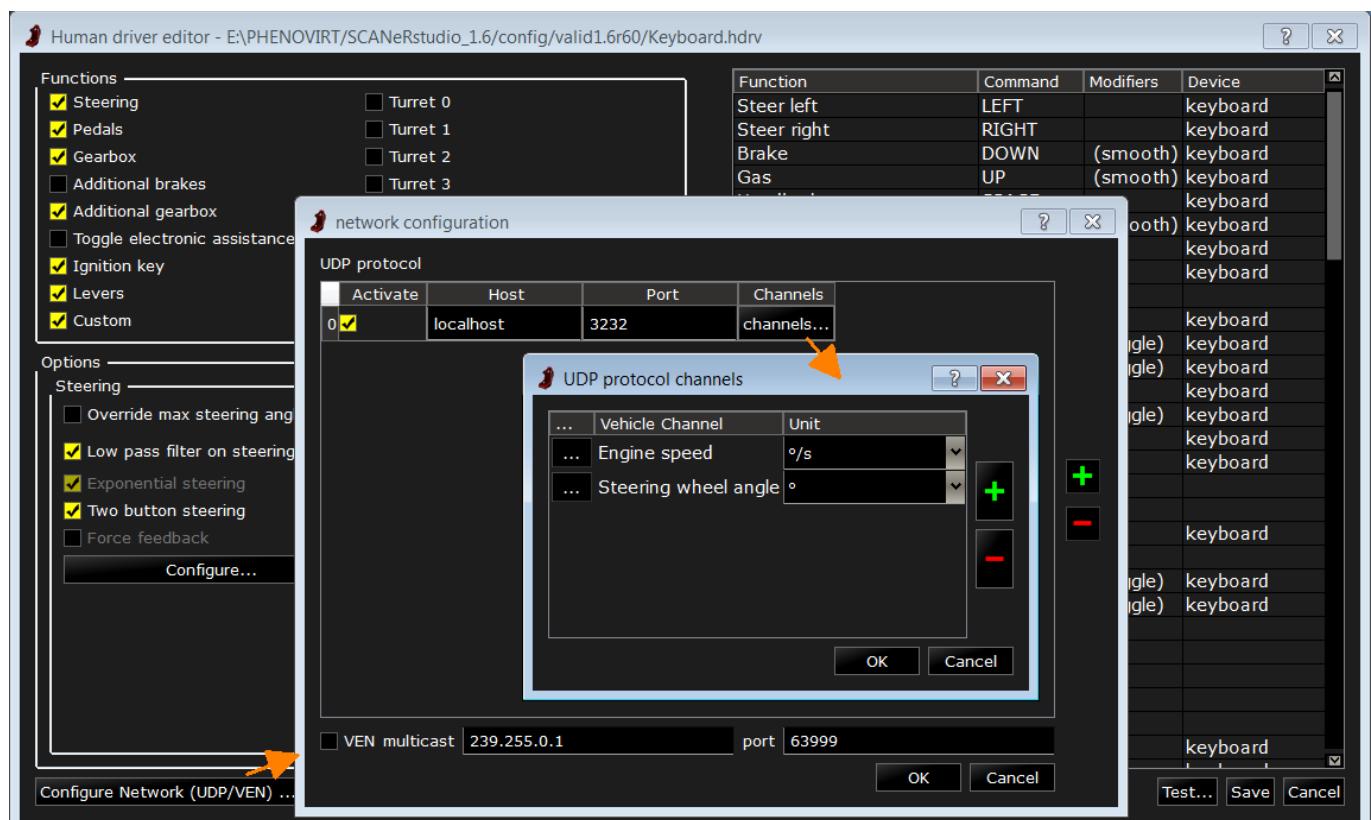


Illustration 128 :

Launch ACQUISITION (DriverHandler.exe and link the HDRV) and the MODELHANDLER.

Run and drive a scenario using a "Human driver (interactive)" vehicle.

Then you can check that the UDPCabin is receiving data:

recieved packet 3070 containing: 2 floats: 13880.3 -134.825
sending packet...

Without stopping the scenario, right-click on the ACQUISITION to select the "Edit process" menu. Then edit the HDRV file. Next double click on any line in the "Function mapping" array. Next select the "joystick..." button:

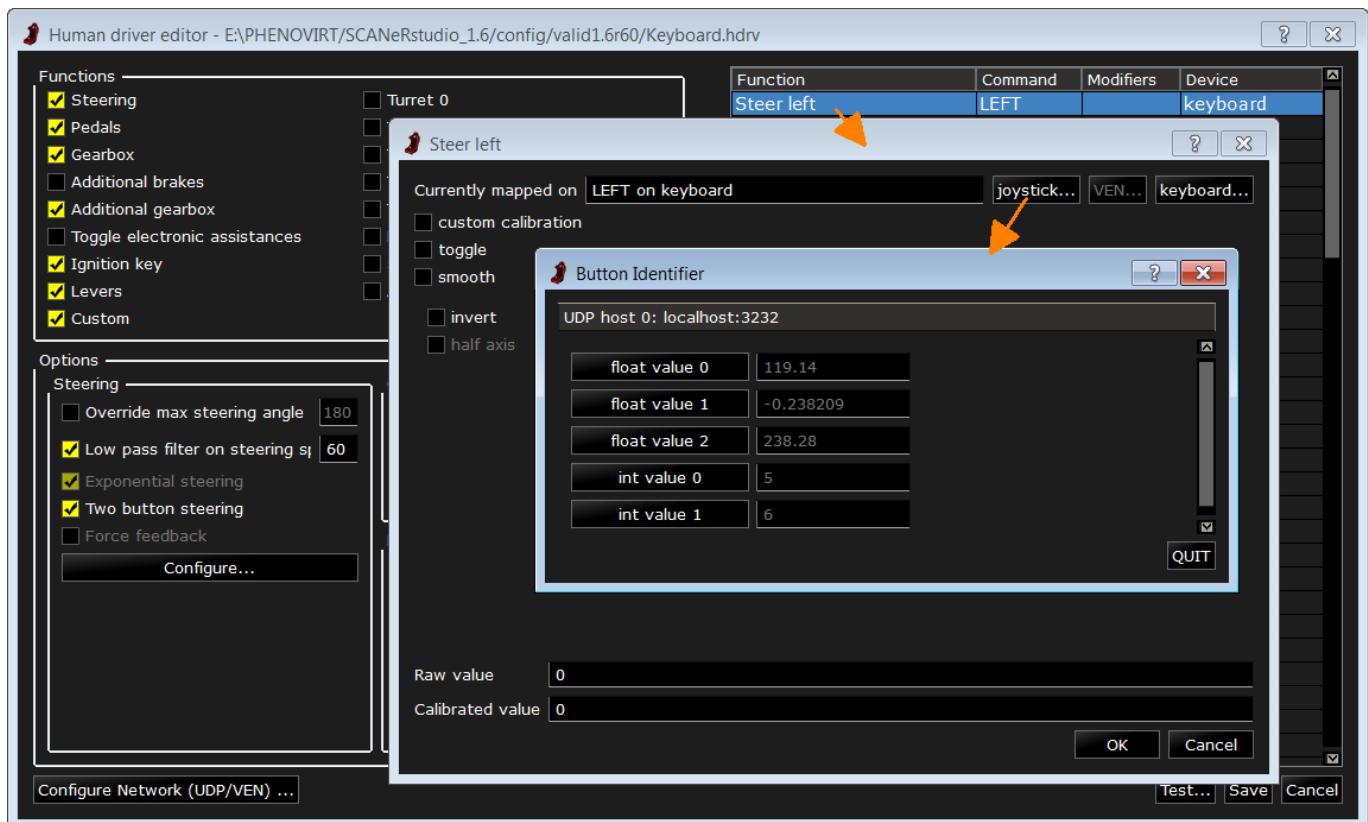


Illustration 129 :

Then you can check in the "Button Identifier" GUI (under the UDP host 0: <PC>:3232 area) that the UDPCabin is sending some data too.

xix - VEN API

xix.a - INTRODUCTION

VEN means "Versatile Extended Network"

It is a versatile communication bus, inspired by the vehicle CAN bus. It does not depends on the SCANeR™ studio environment. It can be used by any software (using a dynamic library: DLL) to send and receive data from other software, including SCANeR™ studio binaries¹⁵.

For example, this bus, via a software created by the user, can:

- Send / receive data on a PC port,
- Send / receive data between heterogeneous software,
- Trigger a script event by using external action.

The dynamic library SCANeR™ studio VEN is delivered in order to open the software to the user applications.

To be compliant with a lot of application, it is delivered in the form of a DLL (which contains all needed methods), with a "C language" binding. It can be mapped with applications written in C, C++, Python or any application able to use a DLL.

xix.a.1 - CHANGES

The VEN interface has changed since the last SCANeR™ studio 1.3 version.

The new interface needs a new parameter at initialization. This parameter determines the way to send a VEN message on network:

- `VENSendNonBuffered` : the VEN message is send immediately on network.
- `VENSendBuffered` : The VEN messages are buffered while the buffer size has not reach its limit (2800 bytes) or the user calls `VENFlush`. Then the buffer is send on the network. Use this mode when the number of VEN messages send in a cycle becomes important. It's a way to decrease the number of network packets.

The VEN.dll from SCANeR™ studio 1.3 versions are no more compatible with VEN.dll from SCANeR™ studio 1.4.

However, it is possible to use a VEN application compiled with the old interface, by copying the VEN.dll delivered with SCANeR™ studio 1.4 in the execution folder of the VEN application.

To use the new VEN interface, the application have to be compiled, with the VEN2 interface :

- use the header file VEN2.h our VEN2CommunicationSystem.h
- link with VEN2.lib
- copy VEN2.dll in the application execution folder

¹⁵: see more in the VEN chapter of the SCANeRstudio_SCENARIO_UserManual documentation.

XIX.B - ENVIRONMENT

Refer to the "Environment" chapter (page 18).

XIX.C - PACKAGE

The SCANeR™ studio API package includes:

- 2 DLL files : **VEN.dll** and **VEN2.dll** (located in
<STUDIO_PATH>\<STUDIO_VERS>\APIs\bin\<PLATFORM>\vs2013) & .
- A LIB files: **VEN.lib** and **VEN2.lib** (located in
<STUDIO_PATH>\<STUDIO_VERS>\APIs\lib\<PLATFORM>\vs2013)
- 4 include files, **VEN.h(C)** & **VENCommunicationSystem.h(C++)** , **VEN2.h(C)** &
VEN2CommunicationSystem.h(C++) (located
<STUDIO_PATH>\<STUDIO_VERS>\APIs\include\VEN)
- This documentation
- A MS Visual C++ 8 solution (Basic) which contains an example written in C.
- A MS Visual C++ 8 solution (Interpreter) which contains an example written in C++.
- Some python examples (basic and scenario).

XIX.D - DESCRIPTION

XIX.D.1 - PRINCIPE

It is a versatile communication bus, inspired by the vehicle CAN bus. It does not depend on the SCANeR™ studio environment. It can be used by any software (using a dynamic library: DLL) to send and receive data from other software, including SCANeR™ studio binaries¹⁶.

For example, this bus, via a software created by the user, can send / receive data between heterogeneous software,

Trigger a script event by using external action.

XIX.D.1.i.a - Messages

The VEN API consider messages as:

<i>A head</i>	<i>SRC</i>	Is a string. It's an arbitrary name used to group messages together into categories.
	<i>DATA</i>	Is a string. It's an arbitrary name which distinguishes the message of this 'message source' from others messages of such 'message source'.
<i>A body</i>	<i>VAL[n]</i>	An array N elements.

Tableau 41: VEN – messages considered from scripting language

To use it, you must first go through an initialization phase.

This initialization can be done either by specifying, in the code, the network parameters .

Added to these simple initialization features there is "advanced" features to use, in a sophisticated way, the library (allowing you to see all messages circulating on the network without knowledge of them).

The message is characterized by a string of the form "SRC/DATA".

A function exists to know if data is available for a given message.

Permanently you can also get the current data transmitted by using a message. These are the last retrieved. The time at which these data were collected is also provided. Next to the "get" step , the message is marked, by the library, as old as message but you can still get it. Its state is set again as "new" when a new message with the same name is read by the network library.

For transmission system is identical to the reception, the only difference is that one has a function for writing data rather than function for sending.

¹⁶: as SCENARIO module, as VENAnalyzer (see chapter X.D.14.iii - page 168 in "SCANeRstudio_SCENARIO_UserManual" documentation) or as your own module developed with SCANeR™ studio API and with VEN API.

A function to release all the resources is available. After calling this function, an initialization phase is again necessary if one wants to reuse the system.

XIX.D.2 - OPERATING MODE

There are 2 operating modes for data receiving; these modes have no effect on sending:

- A basic mode
- An advanced mode

Select one of them at initialization. This choice could not be changed afterwards.

XIX.D.2.i - Sending - Basic

The parameter `sendMode` of the `VENInit` function is set to `VENNOnBuffered`.

The send is immediate, it means that calling `VENMessageWrite` send directly the message.

XIX.D.2.ii - Sending - Advanced

The parameter `sendMode` of the `VENInit` function is set to `VENSendBuffered`.

The messages are buffered while the buffer size has not reach its limit (2800 bytes) or the user calls `VENFlush`. Then the buffer is send on the network.

Use this mode when the number of VEN messages send in a cycle becomes important. It's a way to decrease the number of network packets.

XIX.D.2.iii - Receiving - Basic

When the system receives a data, in basic mode, then the reception and the execution of the program using the library are done at the same time.

Reception is done when the function `VENMessageRead` is called.

Use the `VENMessageAvailable` function to know if there is a new version of the VEN message available (it means if an update occurs since the last read of this message).

XIX.D.2.iv - Receiving - Advanced

When the system receives a data, in advanced mode, then it triggers a callback function. There is no filter ; all messages trigger a callback.

In this mode, functions `VENMessageRead` and `VENMessageAvailable` are disabled (a call to one of these will fail).

Function `VENAdvanceInit` initialize this mode.

Data is received into a callback `VENRecieveCallBack`.

XIX.D.3 - INTERFACE

This section presents the interface that is available. In fact, two interfaces are available: C and C++.

There are 5 main functions:

- Initialization of VEN communication,
- Test to know a VEN message is available,
- Write a VEN message.,
- Read a VEN message.
- Release resources.

The principles of use is:

- Initialize the VEN communication,
- Read and/or write VEN message,
- Release resources.

XIX.D.3.i - Initialization

Condition	Explanation
<p>C functions :</p> <pre>int VENInit(VENCom* com, const char* multicastAddress, unsigned short port, VENSendMode sendMode)</pre> <p>or</p> <pre>int VENAdvanceInit(VENCom* com, const char* multicastAddress, unsigned short port, VENSendMode sendMode, void* callbackData, VENRecieveCallBack callback)</pre> <p>C++ functions :</p> <pre>VENCommunicationSystem :: init(const std::string& multicastAddress, unsigned short port, VENSendMode sendMode, VENRecieveCallBack callback = NULL)</pre>	<p>Initialize the VEN communication system.</p> <ul style="list-style-type: none"> • <code>com</code> is a VEN communication object. • <code>multicastAddress</code> is a string. It set the multicast address. • <code>port</code> is the used port of the multicast address. • <code>sendMode</code> is an enumerated value. It could be: <ul style="list-style-type: none"> ◦ <code>VENSendNonBuffered</code>: the VEN message is sent immediately on network ◦ <code>VENSendBuffered</code>: the VEN message is buffered. The buffer is send on network after a flush • <code>callbackData</code> for incoming message • <code>callback</code> is a callback function to handle any received message <p>Return 0 when succeed</p>

XIX.D.3.ii - Trigger a message reception (only for Advanced mode)

<pre>C functions : typedef void (*VENRecieveCallBack) (void* callbackData, const char* messageName, VENType typeOfData, const void* arrayOfData, unsigned int numberOfElements);</pre>	<p>Say if an unread data of a message is available or not.</p> <ul style="list-style-type: none"> <code>callbackData</code> is the pointer passed to <code>VENAdvancedInit</code>. <code>messageName</code> contains the source and the message name. The source and the message are separated with the char “/”. Format: “SOURCE/MESSAGE”. <code>typeOfData</code> defines the type of data to read. <code>arrayOfData</code> contains all database. <code>numberOfElements</code> provides the number of elements that the <code>arrayOfData</code> contains
--	---

XIX.D.3.iii - Availability test

Condition	Explanation
<pre>C functions : int VENMessageAvailable(VENCom* com, const char* srcName, const char* messageName) or C++ functions : int VENCommunicationSystem ::available(const std::string&srcName, const std::string&messageName)</pre>	<p>Say if an unread data of a message is available or not.</p> <ul style="list-style-type: none"> <code>com</code> is a VEN communication object. <code>srcName</code> is a string that describes the 1st part of the head of the message. <code>messageName</code> is a string that describes the 2nd part of the head of the message. <p>Returns: Returns 1, 0</p> <ul style="list-style-type: none"> ○ 1: if the data is not read yet, ○ 0: if no data or if an already read data is present, ○ other: fault.

Values are readable without "availability test". This test just indicates if an unread message is present or not.

Read a message, without doing such test, returns the last value of the message even if the message has been read yet.

XIX.D.3.iv - Read

Function	Explanation
<p>C functions :</p> <pre>int VENMessageRead(VENCom* com, const char* srcName, const char* messageName, VENType type, void* buffer, const unsigned int elementNumber)</pre> <p>or</p> <p>C++ functions :</p> <pre>int VENCommunicationSystem :: read(const std::string& srcName, const std::string& messageName, VENType type, void* buffer, unsigned int elementNumber)</pre>	<p>Return of the values contained in the VEN message.</p> <ul style="list-style-type: none"> • <code>com</code> is a VEN communication object. • <code>srcName</code> is a string that describes the 1st part of the head of the message. • <code>messageName</code> is a string that describes the 2nd part of the head of the message. • <code>type</code> is the type of the written data (double, int or string). • <code>buffer</code> is an array in which the value of the message is written. • <code>ElementNumber</code> is the number of item in the array. <p>Write the value of the message in a buffer-array. Returns:</p> <ul style="list-style-type: none"> ○ 0: when succeed, ○ 5: when buffer size and data of the message size do not match.

In C, many useful functions are available. They take, as parameters, a set of variables instead of the buffer array.

In C + + functions taking a typed pointer (int *, double or char *) are available.

XIX.D.3.v - Write

Function	Explanation
<p>C functions :</p> <pre>int VENMessageWrite(VENCom* com, const char* srcName, const char* messageName, VENType type, void* data, const unsigned int elementNumber)</pre> <p>or</p> <p>C++ functions :</p> <pre>int VENCommunicationSystem :: write(const std::string& srcName, const std::string& messageName, VENType type, const void* data, unsigned int elementNumber)</pre>	<p>Send decimal numbers into a VEN message</p> <ul style="list-style-type: none"> • <code>com</code> is a VEN communication object. • <code>srcName</code> is a string that describes the 1st part of the head of the message. • <code>messageName</code> is a string that describes the 2nd part of the head of the message. • <code>type</code> is the type of the written data (double, int or string). • <code>data</code> is an array containing data to send. • <code>ElementNumber</code> is the number of item in the array. <p>Return 0 when succeed</p>

In C, many useful functions are available. They take, as parameters, a set of variables instead of the data array.

In C + + functions taking a typed pointer (int *, double or char *) are available.

XIX.D.3.vi - Flush

Function	Explanation
C functions : <pre>int VENFlush()</pre> <p>or</p> C++ functions : <pre>int VENCommunicationSystem ::flush()</pre>	<p>If VEN communication system is initialized with the sending mode set as <code>VENSendBuffered</code>, it need to call the flush function to be sure the buffer is send on network in the same simulation time.</p> <p>The flush function does nothing if the sending mode is set as <code>VENSendImmediate</code></p> <p>An automatic flush is done when the buffer size is reached.</p> <p>Return 0 when succeed</p>

XIX.D.3.vii - Free resources

Function	Explanation
C functions : <pre>int VENRelease(VENCom* com)</pre> <p>or</p> C++ functions : <pre>int VENCommunicationSystem ::release()</pre>	<p>Make allocated resources free.</p> <ul style="list-style-type: none"> • <code>com</code> is a VEN communication object.

If initialization parameters need to be changed then call this function. Reset the system to use it.

XIX.D.4 - LINK WITH SCRIPTING LANGUAGE

Read more about how VEN and scripting language match in the VEN chapter of the SCANeRstudio_SCENARIO_UserManual documentation.

XIX.D.4.i - scenario mode use case

See scenario « VEN.sce ». It contains 3 tasks:

- VEN initialization,
- Receives data from VEN,

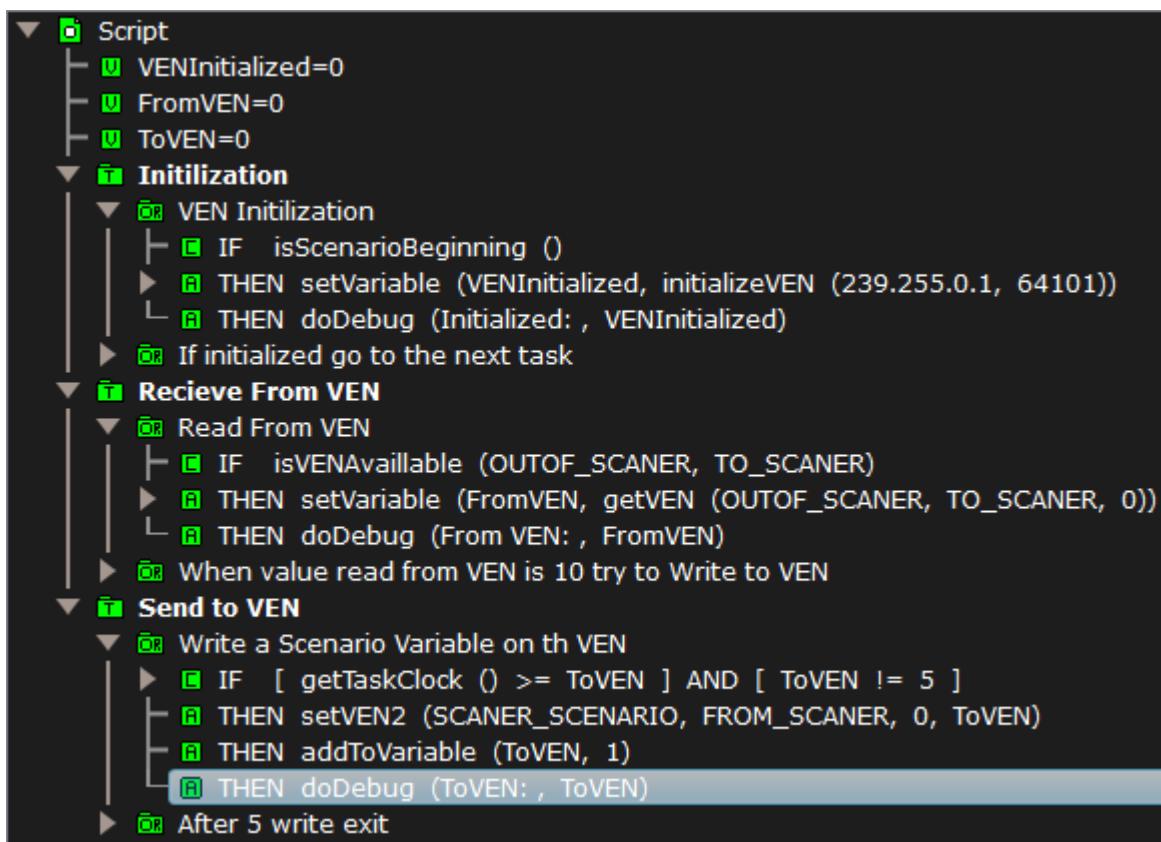


Illustration 130: script of the VEN test scenario

- Sends data to VEN.

XIX.D.4.i.a - Initialization

The task calls the initialization parameters initVEN with multicast port. If this function succeeds (returns a value other than 0) then script jumps to the receiving step (bellow: XIX.D.4.i.b).

XIX.D.4.i.b - Data receiving

This task check if an unread message with « SCANER_SCENARIO » in the 1st part of the head and with a name : « TO_SCANER » exists.

If such message exists, then the system reads it and copy the 1st value in the variable « FromVEN ».

Once this variable is 10, then script jumps to the sending step (below: XIX.D.4.i.c).



If isVENAvailable() is omitted, then the following message is shows in the APPLICATION LOG :
SCENARIO : Nothing to read



Only double data type message are available in the script, due to script limitation. If the value of the body is an integer then the following message is shows in the APPLICATION LOG :
SCENARIO : Bad Type Integer

For example, in the VENAnalyser set 1.0 instead of 1 in the field 'Data'.

XIX.D.4.i.c - Data sending

This task write each second a message with « SCANER_SCENARIO » in the 1st part of the head and with a name : « FROM_SCANER ». It's a message with 2 parameters:

- the 1st on : 0
- The 2nd on: the variable « ToVEN ».

This message is send 5 times. Each send, the variable « ToVEN » is incremented (+1).

XIX.D.5 - BINDING PYTHON

XIX.D.5.i - Methods

The python binding is the C++ interface shown. Just use the VENCommunicationSystem. We describe bellow used methods.

Methods	Explanation
init(multicast_address, port);	<p>Initialize the VEN communication system.</p> <ul style="list-style-type: none"> • <code>MulticastAddress</code> is a string. It set the multicast address. • <code>Port</code> is the used port of the multicast address <p>Return 0 when succeed</p>

Methods	Explanation
<pre>read_int(src_name, msg_name, data, size)</pre> <pre>read_float(src_name, msg_name, data, size)</pre>	<p>Read decimal or integer numbers from VEN message.</p> <ul style="list-style-type: none"> • <code>Src_Name</code> is a string that describes the 1st part of the head of the message. • <code>Msg_Name</code> is a string that describes the 2nd part of the head of the message. • <code>Size</code> is the number of data in the VEN message. • <code>Data</code> is a list in which hidden data is written. <p>Return 0 when succeed</p>
<pre>read_string(src_name, msg_name, string)</pre>	<p>Read a string from VEN message.</p> <ul style="list-style-type: none"> • <code>Src_Name</code> is a string that describes the 1st part of the head of the message. • <code>Msg_Name</code> is a string that describes the 2nd part of the head of the message. • <code>String</code> contains the hidden string. <p>Return 0 when succeed</p>
<pre>write_int(src_name, msg_name, data)</pre> <pre>write_float(src_name, msg_name, data)</pre>	<p>Send decimal or integer numbers into a VEN message</p> <ul style="list-style-type: none"> • <code>Src_Name</code> is a string that describes the 1st part of the head of the message. • <code>Msg_Name</code> is a string that describes the 2nd part of the head of the message. • <code>Data</code> is a list containing data to send. <p>Return 0 when succeed</p>
<pre>write_string(src_name, msg_name, string)</pre>	<p>Read a string into a VEN message.</p> <ul style="list-style-type: none"> • <code>Src_Name</code> is a string that describes the 1st part of the head of the message. • <code>Msg_Name</code> is a string that describes the 2nd part of the head of the message. <p>String contains string to send.</p> <p>Return 0 when succeed</p>

Tableau 42: VEN – Binding Python

XIX.D.5.ii - Example

This example can communicate with the sample script paragraph 3.1.2.

```
import VEN
import time

com = VEN.VENCommunicationSystem()
# VEN initialization

com.init("239.255.0.1", 64101)

# send 11 messages through VEN, one per second
# the message is composed by 2 float values, the first value is incremented by one at each sent
to_scanner_data = [1, 0]
out_sender = "OUTOF_SCANNER"
out_msg = "TO_SCANNER"
for i in range(11):
    time.sleep(1)
    if com.write_float(out_sender, out_msg, to_scanner_data) == 0:
        print 'send ' + out_sender + '/' + out_msg
        print to_scanner_data
    else:
        print com.get_error_string()
    to_scanner_data[0] = to_scanner_data[0] + 1

# receive message from VEN, one read second frequency
# the message is 2 float values, stop reading message when the first message value is 4
in_sender = "SCANER_SCENARIO"
in_msg = "FROM_SCANNER"
data_out = -1
nb_try = 0
while data_out != 4 and nb_try < 100:
    from_scanner_data = []
    time.sleep(1)
    if com.read_float(in_sender, in_msg, from_scanner_data, 2) == 0:
        print 'receive ' + in_sender + '/' + in_msg
        print from_scanner_data
        data_out = from_scanner_data[1]
    else:
        print com.get_error_string()
    nb_try = nb_try + 1
```

Tableau 43

xx - VEN SYNCHRONIZATION

xx.A - INTRODUCTION

The user can start and stop non SCANeR™ modules from the Simulation mode of SCANeR™ studio. A non SCANeR™ module is a Windows executable that doesn't use the SCANeR™ communication protocol to communicate. A non SCANeR™ module can synchronize its state with the simulation state using the VEN protocol.

XX.B - CONFIGURATION

The VEN bus used by SCANeR™ studio to communicate with other modules is configured using the Configuration / VEN Configuration menu. The values entered in this dialog box are stored in the file VEN.cfg, located in the current configuration directory. You can read this file if you want to be sure that your module use the same VEN bus that SCANeR™ studio.

The list of modules participating in a simulation is configured using the Configuration Manager. When you add a process that will be synchronized over VEN, you must choose "VEN" as a module type. The process name will be used by SCANeR™ studio to synchronize your module with the simulation. You must use the process name as the source field of the VEN messages you send to Studio.

xx.c - SYNCHRONIZATION PROTOCOL

SCANeR™ studio continuously sends (roughly every second) a “SimulationState” message on the VEN bus. This message indicates the current simulation state. VEN synchronized modules must reply to this message by sending a “ModuleState” message giving their state. If a module stops replying, SCANeR™ studio removes the state symbol over the module icon to inform the user that the application is not responding.

When the user starts the simulation by pressing the Play button, SCANeR™ studio does the following:

- Send a “RecordState” message on the VEN bus to give the recording mode of the scenario (enabled or disabled)
- Send a “UserData” message on the VEN bus to give the user data of the scenario
- Send a “SimulationState” message with value=LOAD on the VEN bus to request custom modules to load their data
- Wait for every SCANeR™ modules to be ready and for every VEN modules to be in the LOADED state
- Send a “RecordDir” message on the VEN bus to give the recording directory name
- Send a “SimulationState” message with value=INIT on the VEN bus to request custom modules to initialize
- Wait for every SCANeR™ modules to be ready and for every VEN modules to be in the READY state
- Send a “SimulationState” message with value=PLAY on the VEN bus to indicate that simulation is now starting

When the user suspends the simulation by pressing the Pause button, SCANeR™ studio sends a “SimulationState” message with value=PAUSE on the VEN bus.

When the user resumes the simulation by pressing the Play button, SCANeR™ studio sends a “SimulationState” message with value=PLAY on the VEN bus.

When the user stops the simulation by pressing the Stop button, SCANeR™ studio sends a “SimulationState” message with value=STOP on the VEN bus.

A custom module must send on the VEN bus a “ModuleState” message where VEN source matches the module name in the configuration. SCANeR™ studio will then display a state symbol over the module icon:

Sent ModuleState message	Displayed symbol
None	
STOPPED / LOADED / READY	
PLAYING	

Sent ModuleState message	Displayed symbol
PAUSED	

This message must be sent by the module each time it receives a "SimulationState" message.

When the user wants to stop a module by double clicking on its icon in the Simulator status window, SCANeR™ studio sends a "Kill" message with value equals to the module name. The module should handle this message and terminate gracefully. If it doesn't, it will be killed abruptly.

WARNING: when a VEN synchronized module terminates correctly due to a user action (either from SCANeR™ studio, either by another way, for example closing the main window), the module must send a "ModuleState" message with value=-1. It informs SCANeR™ studio that the module is actually stopped and that it must not be waited for when a simulation starts. Forgetting to send this message will prevent the user to start a simulation.

A custom module can send a "PrintInfo", "PrintWarning" or "PrintError" message. SCANeR™ studio will display the message in the "Application Log".

xx.D - MESSAGES REFERENCE

SCANeR™ simulation state

Sending module	SCANeR™ studio
Receiving module	All
Source	Studio
Message	SimulationState
Size	1
Type	Integer
Value	0 : STOP 1 : LOAD 2 : INIT 3 : PLAY 4 : PAUSE

Custom modules state

Sending module	Custom modules willing to be synchronized
Receiving module	SCANeR™ studio
Source	Module name
Message	ModuleState
Size	1
Type	Integer
Value	0 : STOPPED 1 : LOADED 2 : READY 3 : PLAYING 4 : PAUSED

Module termination

Sending module	SCANeR™ studio
Receiving module	All
Source	Studio
Message	Kill
Size	1
Type	String
Value	Name of the module to be killed

Recording mode

Sending module	SCANeR™ studio
Receiving module	All
Source	Studio
Message	RecordState
Size	1
Type	Integer
Value	0 : OFF
	1 : ON

Recording directory

Sending module	SCANeR™ studio
Receiving module	All
Source	Studio
Message	RecordDir
Size	1
Type	String
Value (sample)	C:\OKTAL\SCANeRstudio_1.3\data\USER\record\nomscenario-27_02_2012-16h37m49s

User data

Sending module	SCANeR™ studio
Receiving module	All
Source	Studio
Message	UserData
Size	1
Type	String
Value (sample)	AmbientLight=0.5;Sound=1

Time marker

Sending module	All
Receiving module	SCANeR™ studio
Source	Marker producer
Message	TimeMarker
Size	1
Type	Integer
Value	Unique marker id for this producer

Time marker note

Sending module	All
Receiving module	SCANeR™ studio
Source	Marker producer
Message	TimeMarkerNote_nnn
Size	1
Type	String
Value	Marker note

Message display

Sending module	All
Receiving module	SCANeR™ studio
Source	Module name
Message	PrintInfo / PrintWarning / PrintError
Size	1
Type	String
Value	Text to display

xxi - INDEXES

xxi.A - LEXICAL INDEX

3DS.....	258	Implementation.....	259	Output.....	26	URI.....	130
API.....	130	In bl.....	249	PostDraw.....	259	UTF-8.....	130
Channel.....	259	Include...20, 111, 119, 192, 291		PreDraw.....	259	Vehicle Dynamics ComUdp	205
Config.....	259	Input.....	26	SampleEditableVehicle.h....192		VehicleDynamics.....	192
ControlSFX.....	50	Interface.....	259	SCANeR_API_C.1.X.dll.....20		VehicleDynamicsAdvanced	192
Custom Input.....	193	IVE.....	258	SCANeR_API_C.lib.....20		VehicleDynamicsSample.dll	
Custom Ouput.....	193	IVehicleDynamics.....	193	ScanerAPI_DLL_C.h.....20			
DAE.....	258	Javascript.....	139	Scenario_API.dll.....119		VEN.....	289
Declare.....	26	JSON.....	130	Scenario_API.lib.....119		VEN.dll.....	291
DLL.....20, 119, 153, 186, 192,		LIB.....20, 119, 291		ScenarioAPI_DLL_C.h.....119		VEN.h(C) &	
193, 197, 198, 228, 289, 291,		Load.....	259	Scene 2D.....	263	VENCommunicationSystem.h(
292		Mdl.....	224	Scene 3D.....	263	C++) , VEN2.h(C) &	
DXF.....	258	Module state.....	135	SHM.....	198	VEN2CommunicationSystem.h	
External.....	197	OBJ.....	258	Simulink.....	18	(C++).....	291
FLT.....	258	OKTAL_CallasPlugin.mdl..155		Slblocks.m.....	155	VEN.lib.....	291
GUI.....	130	OpenSceneGraph.....	258	Strategy.dll.....	228	Weather control.....	50
HTML.....	139	OSG.....	258	Studio state.....	136	h.....	111
HTTP.....	130	OSGB.....	258	TMF.....	224		
HyperText Transfer Protocol	12,	OSGT.....	258	Unload.....	259		
130		OSGX.....	258	Update	259		

xxi.B - ILLUSTRATION INDEX

Illustration 1 : The SCANeR™studio API architecture.....	16	Illustration 31 : the PHYSICS debug option to preview dynamic_ground moves.....	77
Illustration 2 : Use Case of the SCANeR™studio API.....	22	Illustration 32 : Simulink - SCANeR communication protocol.	83
Illustration 3 : SCANeR studio API.....	23	Illustration 33: Simulink – Controller block.....	86
Illustration 4 : SCANeR™studio state management.....	24	Illustration 34 : (S) the module is synchronized with Simulink for Cosimulation with the OFFLINESCHEDULER module.....	86
Illustration 5 : SCANeR™studio state management.....	28	Illustration 35 : Enable Barrier for your binary file with the OFFLINESCHEDULER module.....	86
Illustration 6 : SCANeR™studio modules concept.....	32	Illustration 36: Simulink – Launcher block.....	87
Illustration 7: Shared Memory (Shm).....	34	Illustration 37: Simulink - Input block.....	88
Illustration 8: Shared Memory correctives.....	35	Illustration 38: Simulink - Output block.....	88
Illustration 9 : Sample Environment.....	48	Illustration 39: Simulink – SHM block.....	89
Illustration 10 : Launch a SCANeR API sample from Visual Studio.....	49	Illustration 40: Simulink – Network block.....	89
Illustration 11: Weather Control GUI.....	51	Illustration 41: Simulink – block index.....	89
Illustration 12: Weather Control Communication.....	52	Illustration 42: Simulink - event.....	90
Illustration 13: Client Interface ScanerAPI Communication.....	53	Illustration 43: String fields.....	91
Illustration 14 : Export channel 20 & 21 received in Controlpad.....	54	Illustration 44: Simulink - Fixed size array.....	91
Illustration 15: Model SHM ScanerAPI Communication.....	55	Illustration 45: Simulink - Variable size array (Input block).....	92
Illustration 16 : scriptAcquisitionMapping MICE.....	56	Illustration 46: Simulink - Variable size array (Outpu block).....	92
Illustration 17: Script Acquisition Mapping Communication.....	57	Illustration 47: Simulink – Structure (input block).....	93
Illustration 18: ScanerAPI Spy Communication.....	58	Illustration 48: Simulink – Structure (output block).....	94
Illustration 19: ScanerAPI Spy execution Sample.....	58	Illustration 49: Simulink – As a module in configuration.....	95
Illustration 20: Sample Event ScanerAPI Communication.....	60	Illustration 50: Simulink – Controller block.....	95
Illustration 21 : RADARDETECTION module configuration.....	61	Illustration 51: Sequence Diagram for Cosimulation using the Launcher and Controller blocks.....	97
Illustration 22 : Sample of a use case of the script function getRadarDetection.....	62	Illustration 52: Script.....	100
Illustration 23: Radar ScanerAPI - Communication.....	62	Illustration 53: read static array of standard data type.....	102
Illustration 24 : Radar ScanerAPI – How to.....	63	Illustration 54: read dynamic array of structure data type.....	103
Illustration 25 : TrafficTakeOver Workflow.....	64	Illustration 55 : ProcessManager component.....	105
Illustration 26 : SampleMatLabLaserMeter – real time MATLAB processing.....	70	Illustration 56 : Inputs and outputs of packages.....	105
Illustration 27: ExportChannels GUI.....	71	Illustration 57 : RTMaps Input component.....	106
Illustration 28: ObjectsViewer GUI.....	73	Illustration 58 : RTMaps Output component.....	107
Illustration 29: LaserMeterViewer GUI.....	74	Illustration 59 : RTMaps module.....	109
Illustration 30 : DynamicHeightMap.....	76	Illustration 60 : RTMaps Process Manager configuration.....	109
		Illustration 61 : SampleReplayControlPlugin logs.....	118

Illustration 62: Scenario tree option.....	124	Illustration 98 : Build the ComVhcDyn S-function.....	214
Illustration 63: Scenario tree.....	125	Illustration 99 : Build the sfun_rttme S-function.....	214
Illustration 64: Right click on the scenario tree.....	126	Illustration 100 : ADAS with SCANeR API.....	216
Illustration 65: Vehicle position.....	127	Illustration 101 : ADAS with RTGateway.....	217
Illustration 66: Scenario Script element.....	129	Illustration 102: Overview of motion platform strategy.....	218
Illustration 67 : HTTP API - Http control embedded in studio GUI.....	131	Illustration 103: PassThrough Simulink Motion Strategy sample	
			226
Illustration 68 : HTTP API – studio web monitor.....	140	Illustration 104: DirectReplayStrategy Simulink Motion Strategy sample.....	227
Illustration 69 : HTTP API – Analyse web monitor.....	144	Illustration 105: Motion strategy custom inputs.....	228
Illustration 70 : Include environment.....	150	Illustration 106: custom strategy - Simulink.....	228
Illustration 71 : MatLab set path.....	155	Illustration 107: AFS – PlugIn Architecture.....	229
Illustration 72 : Vehicle block.....	156	Illustration 108: AFS - Software architecture.....	230
Illustration 73 : Callas instance, external models tab.....	162	Illustration 109: AFS - Headlight manager GUI.....	231
Illustration 74 : Shared memory communication.....	163	Illustration 110: AFS - Simulink.....	245
Illustration 75 : Network communication.....	163	Illustration 111 : AFS – projector light sources.....	247
Illustration 76 : Browse network.....	163	Illustration 112:.....	252
Illustration 77 : SDK – No external model used.....	184	Illustration 113: NightTestManager Software - simulink executable.....	256
Illustration 78 : SDK – External model used.....	185	Illustration 114 : NightTestManager Software - Strategy.....	256
Illustration 79 : External model selection.....	185	Illustration 115 : NightTestManager Software – Strategy - select	
			257
Illustration 80 : Compatibility error.....	185	Illustration 116 : Plugin architecture.....	259
Illustration 81 : External model checking.....	186	Illustration 117 : Plugin Class diagram.....	260
Illustration 82 : Externalization idea.....	188	Illustration 118 : visual API class diagram.....	262
Illustration 83 : Externalization in SCANeR API.....	189	Illustration 119: Image Sharing API Non-blocking reading.....	269
Illustration 84 : Simulink model.....	189	Illustration 120: Image Sharing API Blocking reading.....	270
Illustration 85 : rtwWrapper vs Co-simulation design differences	190	Illustration 121: ImageSharing Enabling sharing in a camera sensor.....	274
			274
Illustration 87 : rtwWrapper vs Co-simulation design differences	190	Illustration 122 : RT_GATEWAY protocol.....	278
			278
Illustration 88: cats::model::IVehicleDynamics.....	194	Illustration 123 : Network.html messages to be exported on UDP.....	279
Illustration 89: Simulink.....	197	Illustration 124 : Data mapping on UDP packet.....	279
Illustration 90: Ground cutout of the two front wheels.....	203	Illustration 125 : UDP Acquisition Architecture.....	282
Illustration 91 : ComVhcDyn block.....	205	Illustration 126 : Cabin_Acqquisition GUI.....	284
Illustration 92 : ComUdp workflow.....	206	Illustration 127 : NI Measurement & Automation Explorer (NI MAX).....	286
Illustration 93 : Ground reference.....	207	Illustration 128 :.....	287
Illustration 94 : Measurement point.....	207	Illustration 129 :.....	288
Illustration 95 : RemoteVehicleModel sample.....	212	Illustration 130: script of the VEN test scenario.....	298
Illustration 96 : Remote host and Synchronized mode settings	213		
Illustration 97 : Select a compiler using mex -setup command	214		

XXI.C - TABLE INDEX

Table 1: Process controller methods.....	38	Table 22: HTTP API – JSON api/path?name=data.....	137
Table 2: Communication controller methods.....	44	Table 23: HTTP API – JSON api/records.....	137
Table 3: Simulation controller method.....	47	Table 24: HTTP API – JSON api/time.....	137
Tableau 4: Weather Control API.....	50	Table 25: HTTP API – scenario management JSON vs SCANeR™ studio GUI.....	138
Tableau 5: Client Interface ScanerAPI.....	53	Table 26: HTTP API – configuration management JSON vs SCANeR™ studio GUI.....	138
Tableau 6: Model SHM ScanerAPI.....	55	Table 27: HTTP API – scenario management JSON vs	
Tableau 7: Mice Acquisition Mapping ScanerAPI.....	56	AnalysingTool GUI.....	139
Tableau 8: Sample Event ScanerAPI.....	60	Table 28: Outputs of the vehicle model.....	200
Tableau 9: Radar ScanerAPI.....	62	Table 29: motion strategy inputs.....	220
Tableau 10: Sample Simulation ScanerAPI.....	67	XIII.B.1.iii - Table 30: motion strategy outputs.....	222
Tableau 11: ExportChannels ScanerAPI.....	71	Table 31 : AFS – Input data.....	233
Tableau 12: ObjectsViewer ScanerAPI.....	72	Table 32 : AFS - origins of input data.....	236
Tableau 13: LaserMeterViewer ScanerAPI.....	74	Table 33 : AFS – Output data structures.....	237
Table 14: HTTP API - Acronym.....	130	Table 34 : AFS – Input data structures.....	237
Tableau 15: HTTP API - Studio.....	133	Table 35: AFS - Functions description.....	243
Tableau 16: HTTP API - AnalysingTool.....	134	Tableau 36: AFS – SCANeRData Block - Input.....	250
Table 17: HTTP API – JSON api/scenarios.....	134	Tableau 37: AFS – ControlStrategy Block - Output.....	253
Table 18: HTTP API – JSON api/state.....	135	Tableau 38: AFS - ActivationStrategy Block - Output.....	254
Table 19: HTTP API – JSON api/state – module state.....	135	Table 39 : Example of visualPlugin.cfg file.....	264
Table 20 HTTP API – JSON api/state – studio state.....	136		
Table 21: HTTP API – JSON api/configurations.....	136		

Table 40 : Data Cabin_Acquisition exchange on UDP.....	285	Tableau 42: VEN – Binding Python.....	300
Tableau 41: VEN – messages considered from scripting language.....	292	Tableau 43.....	301

xxi.D - TABLE OF CONTENTS

I - Modification History of SDK.....	6
II - Contact.....	7
III - SDK Overview.....	8
<i>III.A - Introduction.....</i>	8
<i>III.B - APIs brief description.....</i>	8
<i>III.C - APIs sample list.....</i>	9
<i>III.C.1 - AFS API.....</i>	9
<i>III.C.2 - Motion API.....</i>	10
<i>III.C.3 - SCANeR API.....</i>	10
<i>III.C.4 - Scenario API.....</i>	12
<i>III.C.5 - HTTP API.....</i>	12
<i>III.C.6 - VehicleDynamics API.....</i>	13
<i>III.C.7 - VehicleDynamicsCallas API.....</i>	13
<i>III.C.8 - UPD API.....</i>	14
<i>III.C.9 - VEN API.....</i>	14
<i>III.C.10 - VisualPlugin API.....</i>	14
<i>III.C.11 - ImageSharing API.....</i>	14
IV - SCANeR API.....	16
<i>IV.A - Introduction.....</i>	16
<i>IV.A.1 - Advice.....</i>	17
<i>IV.B - Environment.....</i>	18
<i>IV.B.1 - Operating system.....</i>	18
<i>IV.B.2 - Compiler.....</i>	18
<i>IV.B.3 - Third party libraries.....</i>	18
<i>IV.B.3.i - Qt.....</i>	18
<i>IV.B.3.ii - OSG.....</i>	18
<i>IV.B.3.iii - Matlab/Simulink.....</i>	18
<i>IV.B.3.iv - OpenCV.....</i>	18
<i>IV.B.3.v - RTMaps.....</i>	18
<i>IV.B.3.vi - Python.....</i>	19
<i>IV.C - Package.....</i>	20
<i>IV.D - Samples compilation.....</i>	21
<i>IV.E - Aim.....</i>	22
<i>IV.F - Interface of the SCANeR studio API.....</i>	23

IV.F.1 - What is a SCANeR studio module.....	24
/IV.F.1.i - State management of the new module.....	24
/IV.F.1.ii - How a SCANeR Module is started.....	24
IV.F.2 - The Process controller.....	25
IV.F.3 - The Communication controller.....	26
/IV.F.3.i - Introduction.....	26
/IV.F.3.ii - Event system.....	27
IV.F.3.ii.a - SCANeR™ studio States management using events.....	28
/IV.F.3.iii - Network part: Architecture.....	31
IV.F.3.iii.a - Structure.....	32
IV.F.3.iii.b - Example.....	33
IV.F.3.iii.c - Network messages with arrays.....	33
/IV.F.3.iv - SHM part: Architecture.....	34
IV.F.3.iv.a - The data Shared Memory.....	34
IV.F.3.iv.b - Vehicle Dynamics Model Input / Output.....	34
IV.F.3.iv.c - Overload the data from Shared Memory.....	35
IV.F.4 - The Simulation control.....	36
IV.G - ANNEXE 1: controller methods.....	37
IV.G.1 - Process controller methods.....	37
IV.G.2 - Communication controller methods.....	38
IV.H - Simulation controller method.....	45
IV.I - ANNEX 2: Lists of data.....	48
IV.I.1 - Network messages.....	48
IV.I.2 - Shared Memory.....	48
IV.J - ANNEXE 3: SAMPLE MODULES IN SCANeR™ studio ENVIRONMENT.....	48
IV.J.1 - Introduction.....	48
IV.J.2 - Delivered samples description.....	50
/IV.J.2.i - SCANeRAPI SAMPLE PROCESS.....	50
/IV.J.2.ii - SCANeRAPI WEATHERCONTROL.....	50
/IV.J.2.iii - SCANeRAPI CLIENT INTERFACE.....	52
/IV.J.2.iv - SCANeRAPI MODEL SHM.....	55
/IV.J.2.v - SCANeRAPI SCRIPTACQUISITIONMAPPING.....	56
/IV.J.2.vi - SCANeRAPI SPY.....	58
/IV.J.2.vi.a - Use case.....	58
/IV.J.2.vii - SCANeRAPI SAMPLE EVENT.....	60
/IV.J.2.viii - SCANeRAPI RADARDETECTION.....	61
/IV.J.2.ix - SCANeRAPI RADAR.....	62
/IV.J.2.x - SCANeR API TrafficTakeOver.....	64
/IV.J.2.xi - SCANeR API ADAS samples.....	65
/IV.J.2.xi.a - Radar sensor.....	65
/IV.J.2.xi.b - Ultrasonic sensor.....	65
/IV.J.2.xi.c - Camera sensor.....	65
/IV.J.2.xii - SCANeRAPI SAMPLE SIMULATON.....	67
/IV.J.2.xiii - SCANeRAPI MATLAB.....	68
/IV.J.2.xiv - SampleMatLabLaserMeter.....	69

<i>IV.J.2.xv - ExportChannels sample.....</i>	71
<i>IV.J.2.xvi - ObjectsViewer sample.....</i>	72
<i>IV.J.2.xvii - LaserMeterViewer sample.....</i>	74
<i>IV.J.2.xviii - SampleDynamicHeightMap.....</i>	76
<i>IV.J.2.xix - Python samples.....</i>	78
IV.K - Porting from old version.....	79
IV.K.1 - Porting SCANeR API from 1.4 to 1.5.....	79
<i>IV.K.1.i - Network.....</i>	79
IV.K.2 - Porting from SCANeR API 1.3 to 1.5.....	80
<i>IV.K.2.i - SHM.....</i>	80
<i>IV.K.2.ii - Network.....</i>	81
V - SCANeR API with Simulink.....	82
V.A - Introduction.....	82
V.B - Architecture.....	83
V.C - First steps.....	83
V.C.1 - Environment setup.....	83
<i>V.C.1.i - Matlab path.....</i>	83
<i>V.C.1.ii - Compiler configuration.....</i>	84
V.C.2 - Library layout.....	84
V.C.3 - Simulink blocks.....	84
<i>V.C.3.i - Controller.....</i>	84
<i>V.C.3.ii - Inputs / outputs.....</i>	84
<i>V.C.3.iii - Structures.....</i>	85
V.C.4 - Compilation targets.....	85
V.D - Model authoring.....	85
V.D.1 - Compilation target.....	85
V.D.2 - Controller.....	86
V.D.3 - Launcher.....	87
V.D.4 - Network and SHM input blocks.....	87
V.D.5 - Network and SHM output blocks.....	88
V.D.6 - SHM blocks.....	88
V.D.7 - Network blocks.....	89
<i>V.D.7.i - Indexed messages.....</i>	89
<i>V.D.7.ii - Using events for input blocks.....</i>	90
<i>V.D.7.iii - String fields.....</i>	90
V.D.8 - Uniqueness.....	91
V.D.9 - Arrays and structures.....	91
<i>V.D.9.i - Fixed size arrays.....</i>	91
<i>V.D.9.ii - Variable size arrays.....</i>	92
<i>V.D.9.iii - Structures.....</i>	93
V.E - Co-simulation.....	95

V.E.1 - SCANeR™ studio configuration.....	95
V.E.2 - Simulink model configuration.....	95
V.E.3 - Starting and stopping the simulation.....	95
V.E.4 - Using the Launcher Block.....	96
V.E.5 - Sequence Diagram.....	97
V.F - Code generation.....	98
V.F.1 - Target choice.....	98
V.F.2 - Compilation.....	98
V.F.3 - Compilation without Studio.....	98
V.F.4 - Running.....	98
V.G - Samples.....	99
V.G.1 - APIControl.....	99
V.G.2 - LaneKeeping.....	99
V.G.3 - RVLV.....	100
V.G.4 - SpeedLimiter.....	101
V.G.5 - SCANeR_API_DataArray.....	102
VI - SCANeR API with RTMaps.....	104
VI.A - Introduction.....	104
VI.B - First steps.....	104
VI.B.1 - Environment setup.....	104
VI.B.2 - Installation of SCANeR API RTMaps packages.....	104
VI.B.3 - RTMaps packages.....	105
VI.B.3.i - ProcessManager.....	105
VI.B.3.ii - Inputs / outputs packages.....	105
VI.C - Diagram authoring.....	106
VI.C.1 - Network and SHM input components.....	106
VI.C.2 - Network and SHM output components.....	107
VI.C.3 - SHM components.....	107
VI.C.4 - Network components.....	107
VI.C.5 - Arrays and structures.....	107
VI.C.5.i - Fixed size arrays.....	108
VI.C.5.ii - Variable size arrays.....	108
VI.C.5.iii - Structure arrays.....	108
VI.D - Co-simulation.....	108
VI.D.1 - SCANeR™ studio configuration.....	108
VI.D.2 - RTMaps diagram configuration.....	108
VI.D.3 - Starting and stopping the simulation.....	109
VI.E - Limitations.....	110

VII - Analysing tool REPLAY CONTROL API.....	111
<i>VII.A - Introduction.....</i>	111
<i>VII.B - Environment.....</i>	111
<i>VII.C - Package.....</i>	111
<i>VII.D - Samples compilation.....</i>	111
<i>VII.E - Replay control plugin interface.....</i>	112
VII.E.1 - Methods.....	112
<i>VII.F - Delivered samples description.....</i>	118
VII.F.1 - SampleReplayControlPlugin.....	118
VIII - Scenario API.....	119
<i>VIII.A - Presentation of API.....</i>	119
VIII.A.1 - Introduction.....	119
VIII.A.2 - Environment.....	119
VIII.A.3 - Package.....	119
VIII.A.4 - Samples compilation.....	121
VIII.A.5 - Interface of the Scenario API studio API.....	122
<i>VIII.A.5.i - Scenario initialisation.....</i>	122
<i>VIII.A.5.ii - Quit scenario API.....</i>	122
<i>VIII.A.5.iii - Load a scenario.....</i>	122
<i>VIII.A.5.iv - Save a scenario.....</i>	122
<i>VIII.A.5.v - Modify an element of a scenario.....</i>	122
<i>VIII.A.5.vi - Example.....</i>	122
<i>VIII.B - Use in SCANeR studio.....</i>	124
<i>VIII.C - Scenario special elements.....</i>	127
VIII.C.1 - Vehicle.....	127
VIII.C.1.i.a - Vehicle unique identifier.....	127
VIII.C.1.i.b - Vehicle position.....	127
VIII.C.2 - Driver.....	128
VIII.C.3 - Scenario Script.....	129
VIII.C.4 - Variables of a scenario script.....	129
IX - HTTP API.....	130
<i>IX.A - Introduction.....</i>	130
IX.A.1 - Terminology.....	130
IX.A.2 - Presentation.....	130
IX.A.3 - Overview.....	130
IX.A.4 - API description.....	131
<i>IX.A.4.i - Studio HTTP API.....</i>	133
<i>IX.A.4.ii - AnalyzingTool HTTP API.....</i>	134

<u>IX.A.5 - JSON</u>	134
<u>IX.A.5.i - api/scenarios</u>	134
<u>IX.A.5.ii - api/state</u>	135
<u>IX.A.5.iii - api/configurations</u>	136
<u>IX.A.5.iv - api/path?name=data</u>	137
<u>IX.A.5.v - api/records</u>	137
<u>IX.A.5.vi - api/time</u>	137
<u>IX.A.5.vii - Correspondence with SCANeR™ studio GUI</u>	138
<u>IX.B - Samples</u>	139
<u>IX.B.1 - Simple Studio controller HTML5 sample</u>	139
<u>IX.B.2 - Simple Analysis controller HTML5 sample</u>	143
X - Vehicle Dynamics Callas API	146
<u>X.A - Presentation</u>	146
<u>X.B - Definition</u>	146
<u>X.B.1 - Model Types</u>	146
<u>X.B.1.i - Organic model</u>	146
<u>X.B.1.ii - Functional model</u>	146
<u>X.B.2 - Model functions</u>	147
<u>X.B.3 - Model channels</u>	147
<u>X.B.4 - Model Members</u>	147
<u>X.B.4.i - Table Index Conventions</u>	148
<u>X.C - How it works</u>	149
<u>X.C.1 - Initialisation</u>	149
<u>X.C.2 - Computation loop</u>	149
<u>X.D - Vehicle Dynamics Callas API with C++</u>	149
<u>X.D.1 - Introduction</u>	149
<u>X.D.2 - Setting up the development environment</u>	150
<u>X.D.3 - Samples compilation</u>	151
<u>X.D.4 - Make a new external project</u>	151
<u>X.D.4.i - Copy an example</u>	151
<u>X.D.4.ii - Rename the project</u>	151
<u>X.D.4.iii - Implement your new functions</u>	151
<u>X.D.4.iv - Test often</u>	151
<u>X.D.5 - Function call order</u>	152
<u>X.D.6 - Using the external model</u>	153
<u>X.D.6.i - Setting up the external model</u>	153
<u>X.D.6.ii - The configuration file</u>	153
<u>X.D.6.iii - Launching a configuration</u>	153
<u>X.D.6.iv - Debugging</u>	153
<u>X.D.7 - Maintaining an external model</u>	153
<u>X.D.7.i - Changing SCANeR™ studio release numbers</u>	153
<u>X.D.7.ii - Changing SCANeR™ studio versions</u>	154

<u>X.D.7.iii - Compatibility.....</u>	154
<u>X.D.8 - Tips and tricks.....</u>	154
X.E - Vehicle Dynamics Callas API with MatLab\Simulink.....	154
<u>X.E.1 - Introduction.....</u>	154
<u>X.E.2 - First steps.....</u>	154
<u>X.E.2.i - MatLab path.....</u>	154
<u>X.E.2.ii - Compiler configuration.....</u>	155
<u>X.E.3 - Simulink blocks.....</u>	155
<u>X.E.3.i - Inputs.....</u>	155
<u>X.E.3.ii - Output.....</u>	156
X.E.3.ii.a - Vehicle.....	156
<u>X.E.3.iii - Simulink library.....</u>	157
X.E.3.iii.a - Inputs_Blocks.....	158
X.E.3.iii.b - Functioning synthesis.....	159
<u>X.E.4 - Co-simulation.....</u>	159
<u>X.E.4.i - Presentation.....</u>	159
<u>X.E.4.ii - Data exchange methods.....</u>	160
<u>X.E.4.iii - Limitations.....</u>	161
<u>X.E.4.iv - Run a Co-simulation.....</u>	161
X.E.4.iv.a - Vehicle dynamics callas settings.....	161
X.E.4.iv.b - MatLab\Simulink settings.....	164
<u>X.E.4.v - Display results in SCANeR™ studio.....</u>	166
<u>X.E.5 - Vehicle model block inputs.....</u>	167
<u>X.E.5.i - Size of input functions.....</u>	167
X.E.5.i.a - Generalities.....	167
X.E.5.i.b - Recommended Simulink settings.....	168
X.E.5.i.c - Examples.....	169
<u>X.E.5.i.c.1 - Engine.....</u>	169
<u>X.E.5.i.c.2 - Tyre Grip.....</u>	169
<u>X.E.5.i.c.3 - Suspension – Elasticity.....</u>	171
<u>X.E.5.ii - Order of inputs and outputs.....</u>	175
<u>X.E.5.iii - Setting up a simulation in MatLab/Simulink.....</u>	178
X.E.5.iii.a - Numerical method.....	178
X.E.5.iii.b - Diagnostic options.....	178
X.E.5.iii.c - Advanced settings.....	179
X.F - Callas Simulink sub-model (rtw).....	180
<u>X.F.1 - Overview.....</u>	180
<u>X.F.2 - General concept.....</u>	180
<u>X.F.3 - Building a Simulink vehicle sub-system sample.....</u>	180
<u>X.F.4 - Creating a custom vehicle sub-system in Simulink.....</u>	182
X.G - Use an external model on a Vehicle Dynamics Callas.....	183
<u>X.G.1 - Presentation.....</u>	184
<u>X.G.1.i - Checking.....</u>	186
X.G.1.i.a - Externalized Functions.....	186
X.G.1.i.b - Model Options.....	187
X.H - Glossary.....	187

X.I - ANNEXE 1: VehicleDynamicsCalls Samples in SCANeR™ studio environment.....	188
X.I.1 - ExternalDefects sample.....	188
X.I.2 - Matlab_ExternalDefectst sample.....	188
X.I.3 - SimpleEngine sample.....	188
X.I.4 - MatlabSimpleEngine sample.....	188
X.I.5 - MatlabSimpleEngine_rtwWrapper sample.....	189
X.I.6 - SinusPilot sample.....	191
X.I.7 - MatlabSinusPilot sample.....	191
X.I.8 - SinusPilot_C.....	191
XI - Vehicle Dynamics API.....	192
XI.A - Building a vehicle dynamic sample.....	192
XI.A.1 - Package.....	192
XI.A.2 - C++ build environment.....	192
XI.A.3 - Base factory interface.....	193
XI.A.4 - Behaviour description.....	193
XI.A.4.i - Model implementation.....	193
XI.A.4.ii - IVehicleDynamics.....	194
XI.B - Building a Simulink vehicle sample.....	197
XI.B.1 - Simulink build environment.....	197
XI.B.2 - Using the External vehicle model.....	197
XI.B.3 - Inputs and outputs.....	198
XI.B.4 - Interaction with the ground.....	200
XI.B.4.i - Picking the ground.....	203
XI.B.4.ii - Ground cutout.....	203
XII - Vehicle Dynamics ComUDP API.....	205
XII.A - Presentation.....	205
XII.B - Workflow.....	206
XII.C - Reference frames.....	207
XII.D - Vehicle model inputs (driver commands).....	208
XII.E - Vehicle model outputs (Vehicle state).....	209
XII.F - Sample “RemoteVehicleModel”.....	211
XII.F.1 - Presentation.....	211
XII.F.2 - Build and run the sample.....	213
XII.F.2.i - Setup SCANeR™ studio.....	213
XII.F.2.ii - Setup the RemoteVehicleModel Simulink model.....	214
XII.F.2.iii - Run a Simulation.....	215
XII.G - Run my vehicle model in real-time or in non real-time.....	215

XII.G.1 - Real-time in non synchronized mode.....	215
XII.G.2 - Real-time in synchronized mode.....	215
XII.G.3 - Real-time in synchronized mode with an external scheduling.....	215
XII.G.4 - Non Real-time in synchronized mode.....	216
XII.H - How to run an ADAS into my vehicle model?.....	216
XII.H.1 - Use the SCANeR API.....	216
XII.H.1.i - Workflow.....	216
XII.H.2 - Use the RTGateway module.....	217
XII.H.2.i - Workflow.....	217
XIII - MOTION API.....	218
XIII.A - API Presentation.....	218
XIII.B - Description.....	219
XIII.B.1 - Interface.....	219
XIII.B.1.i - Inputs.....	219
XIII.B.1.ii - Outputs.....	220
XIII.B.1.iii - Table 30: motion strategy outputs.....	222
XIII.B.2 - Types of strategies.....	222
XIII.B.2.i.a - Modulation strategies.....	222
XIII.B.2.i.b - Position control strategies.....	222
XIII.B.3 - The C API.....	222
XIII.B.3.i - Principle.....	222
XIII.B.3.ii - Compilation environment.....	223
XIII.B.3.iii - Functions.....	223
XIII.B.3.iii.a - Constructor.....	223
XIII.B.3.iii.b - Compute.....	223
XIII.B.3.iii.c - Destructor.....	224
XIII.B.4 - The Simulink API.....	224
XIII.B.4.i - Using the template make file.....	224
XIII.B.4.ii - Simulink version.....	224
XIII.B.4.iii - C runtime version.....	224
XIII.B.4.iv - valid motion strategy model.....	225
XIII.B.5 - Samples.....	225
XIII.B.5.i - MotionSampleStrategy.....	225
XIII.B.5.ii - PassThrough.....	225
XIII.B.5.iii - DirectReplay.....	226
XIII.B.6 - Using custom strategies with SCANeR™ studio.....	228
XIV - AFS Interface.....	229
XIV.A - Introduction.....	229
XIV.B - Architecture overview.....	229
XIV.C - Input/Output data definitions.....	231
XIV.C.1 - Input data definitions (to the DLL).....	231

XIV.C.2 - Input data origins (to the DLL).....	235
XIV.C.3 - Output data definitions (from the DLL).....	237
XIV.C.4 - Input parameters definition.....	237
XIV.D - Developer guide.....	238
XIV.D.1 - Overview.....	238
XIV.D.2 - Definition of the DLL interface.....	238
XIV.D.3 - Examples.....	243
XIV.D.4 - Platform.....	244
XIV.D.5 - License Management.....	244
XIV.D.6 - AFS API for Simulink.....	244
<i>XIV.D.6.i - Package.....</i>	244
<i>XIV.D.6.ii - Glossary.....</i>	244
<i>XIV.D.6.iii - Principles.....</i>	244
<i>XIV.D.6.iv - AFS strategy edition.....</i>	248
XIV.D.6.iv.a - In Matlab Simulink.....	248
<i>XIV.D.6.iv.a.1 : Opening Matlab.....</i>	248
<i>XIV.D.6.iv.a.2 : Editing the Simulink Model.....</i>	248
<i>XIV.D.6.iv.a.3 : Generating the executable file.....</i>	255
<i>XIV.D.6.iv.a.4 : Copying the executable file In the data directory.....</i>	255
XIV.D.6.iv.b - In NightTestManager Software.....	256
XIV.D.6.iv.c - Running a simulation	257
XV - VISUAL Plugin API.....	258
XV.A - Introduction.....	258
XV.A.1 - Requirements.....	258
XV.A.2 - Plug-in architecture Overview.....	259
XV.B - Plugin Development.....	260
XV.B.1 - Plug-in interface.....	260
XV.B.2 - Plug-in implementation.....	261
XV.B.3 - Accessing visual representation using API functions.....	262
<i>XV.B.3.i - Graphical context.....</i>	262
<i>XV.B.3.ii - Simulation context.....</i>	263
XV.B.4 - Plug-in Examples.....	264
XV.C - Installation notes.....	264
XV.C.1 - Compilation under Windows Vista/7.....	264
XV.C.2 - Plug-in Installation:.....	264
XV.D - API functions Reference.....	265
XV.D.1 - Graphical Context methods.....	265
XV.D.2 - Simulation Context methods.....	265
XV.D.3 - Simulation Infrastructure methods.....	265
XV.D.4 - Simulation Vehicles methods.....	266
XV.D.5 - SimulationWeather.....	266

XVI - Image sharing API.....	268
XVI.A - API Presentation.....	268
XVI.B - How does it work.....	268
XVI.B.1 - Non-blocking reading mode.....	268
XVI.B.2 - Blocking reading mode.....	269
XVI.C - API Description.....	270
XVI.C.1 - Enumerations.....	270
XVI.C.2 - Structures.....	271
XVI.C.3 - Functions.....	271
XVI.D - Streaming from Visual and Camera Sensor.....	273
XVI.D.1 - Visual.....	273
XVI.D.2 - CameraSensor.....	273
XVI.D.3 - Conventions.....	274
XVI.D.4 - Example.....	275
XVI.D.5 - IS_VideoCapture sample descrition.....	276
XVI.E - Limitations and constraints.....	277
XVII - UDP RTGateway protocol.....	278
XVII.A - Introduction.....	278
XVII.B - Protocol.....	278
XVII.B.1 - Socket initialisation.....	278
XVII.B.2 - Recieve packet format.....	278
XVII.B.3 - Send packet format.....	279
XVII.B.4 - Expected behavior.....	279
XVII.C - SCANeR setup.....	279
XVII.D - UDPRTGatewayIO sample.....	280
XVII.D.1 - Environment.....	280
XVII.D.2 - Behaviour.....	280
XVII.D.3 - Compiling.....	280
XVII.D.4 - Testing.....	280
XVII.D.5 - Modifying.....	281
XVIII - UDP Acquisition protocol.....	282
XVIII.A - Introduction.....	282
XVIII.B - Protocol.....	282
XVIII.B.1 - Socket initialisation.....	282
XVIII.B.2 - Recieve packet format.....	282

XVIII.B.3 - Send packet format.....	283
XVIII.B.4 - Expected behavior.....	283
XVIII.C - SCANeR setup.....	283
XVIII.D - UDPCabin sample.....	283
XVIII.D.1 - Behaviour.....	283
XVIII.D.2 - Compiling.....	283
XVIII.D.3 - Testing.....	283
XVIII.D.4 - Modifying.....	284
XVIII.E - UDP_Cabin_Acquisition sample.....	284
XVIII.E.1 - Description.....	284
XVIII.E.2 - Hardware.....	285
XVIII.E.3 - Configuration file.....	286
XVIII.E.4 - Compiling.....	286
XVIII.E.5 - Testing.....	287
XIX - VEN API.....	289
XIX.A - Introduction.....	289
XIX.A.1 - CHANGES.....	289
XIX.B - Environment.....	290
XIX.C - Package.....	291
XIX.D - Description.....	292
XIX.D.1 - Principe	292
XIX.D.1.i.a - Messages.....	292
XIX.D.2 - Operating Mode.....	293
XIX.D.2.i - Sending - Basic.....	293
XIX.D.2.ii - Sending - Advanced.....	293
XIX.D.2.iii - Receiving - Basic.....	293
XIX.D.2.iv - Receiving - Advanced.....	293
XIX.D.3 - Interface.....	294
XIX.D.3.i - Initialization.....	294
XIX.D.3.ii - Trigger a message reception (only for Advanced mode).....	295
XIX.D.3.iii - Availability test.....	295
XIX.D.3.iv - Read.....	296
XIX.D.3.v - Write.....	296
XIX.D.3.vi - Flush.....	297
XIX.D.3.vii - Free resources.....	297
XIX.D.4 - Link with scripting language.....	298
XIX.D.4.i - scenario mode use case.....	298
XIX.D.4.i.a - Initialization.....	298
XIX.D.4.i.b - Data receiving.....	299
XIX.D.4.i.c - Data sending.....	299
XIX.D.5 - Binding Python.....	299

<u>XIX.D.5.i - Methods.....</u>	299
<u>XIX.D.5.ii - Example.....</u>	301
<u>XX - VEN Synchronization.....</u>	302
<u>XX.A - Introduction.....</u>	302
<u>XX.B - Configuration.....</u>	303
<u>XX.C - Synchronization protocol.....</u>	304
<u>XX.D - Messages reference.....</u>	306
<u>XXI - Indexes.....</u>	310
<u>XXI.A - Lexical index.....</u>	310
<u>XXI.B - Illustration index.....</u>	310
<u>XXI.C - Table index.....</u>	311
<u>XXI.D - Table of contents.....</u>	313
<u>XXII - Appendix.....</u>	326

xxII - APPENDIX