TABLE OF CONTENTS

1. EVAL_ADAS - Develop an ADAS system

This configuration demonstrates how to control an EGO vehicle with an ADAS system.

The scenario to use with this configuration is "**EVAL_ADAS_CTRL**".

In this scenario, the EGO vehicle is controlled by a virtual driver (driver commands emulator).

The EGO vehicle goes straight, it is equipped with a radar sensor, when it comes closer of the pedestrian the pedestrian will cross the road without paying attention to the EGO vehicle, using the SCANeR™ API[i] a basic ADAS system will force the EGO vehicle to brake when the pedestrian is closer than 20 meters in the radar sensor referential.

The source code of the basic ADAS system developed with the SCANeR™ API is delivered into the evaluation data package under "…\APIs\samples\EVALUATION_APIs\scanerAPI\EVAL_LONGITUDINAL_CTRL\"

- C/C++: eval_longitudinal_ctrl_pedalPos.cpp
- Python: eval_longitudinal_ctrl_pedalPos.py
- Simulink: eval_longitudinal_ctrl_pedalPos.slx

2. EVAL_LATERAL_CTRL - Manage the lateral command of your vehicle

This configuration demonstrates how to control the steering wheel of an EGO vehicle depending of the road markings.

The scenario to use with this configuration is "**EVAL_LATERAL_CTRL**".

In this scenario, the human driver only controls the longitudinal part of the EGO vehicle.

The steering wheel of the EGO vehicle follows the road curvature. To detect the road curvature a camera sensor for road markings detection is used.

The source code of the basic ADAS system developed with the SCANeR™ API is delivered into the evaluation data package under "…\APIs\samples\EVALUATION_APIs\scanerAPI\EVAL_LATERAL_CTRL**eval_lateral_ctrl.cpp**".

3. EVAL_LONGI_CTRL - Manage the longitudinal commands of your vehicle

This configuration demonstrates how to control the longitudinal part of an EGO vehicle.

With SCANeR™studio 3 methods are available for longitudinal control:

- Speed,
- or Acceleration,
- or Pedals position.

The scenario to use with this configuration is "**EVAL_LONGI_CTRL**".

In this scenario, the human driver controls the lateral part of the EGO vehicle.

The longitudinal of the EGO vehicle follows SCANeR™ API[i] information.

The EGO vehicle goes straight by default, depending of the selected control here are the expected behaviours:

- Speed: The EGO vehicle will try to reach 100 km/h.
- or Acceleration: The EGO vehicle will …
  - from 0s to 10s: The vehicle accelerates at 1.1 m/s^2.
  - from 10s to 20s the vehicle does not accelerate.
  - from 20s to the end the vehicle decelerates (brake).
- or Pedal position: The EGO vehicle will …
  - from 0s to 10s: The vehicle accelerates at maximum.
  - from 10s to 20s the vehicle does not accelerate.
  - from 20s to the end the vehicle brakes.

The source codes are delivered into the evaluation data package under "…\APIs\samples\EVALUATION_APIs\scanerAPI\EVAL_LONGITUDINAL_CTRL\"

- Speed:
  - C/C++: eval_longitudinal_ctrl_speed.cpp
  - Python: eval_longitudinal_ctrl_speed.py
  - Simulink: eval_longitudinal_ctrl_speed.slx
- Acceleration:
  - C/C++: eval_longitudinal_ctrl_accel.cpp
  - Python: eval_longitudinal_ctrl_accel.py
  - Simulink: eval_longitudinal_ctrl_accel.slx
- Pedal position
  - Python: eval_longitudinal_ctrl_pedalPos.py
  - *See also the **EVAL_ADAS** sample*

4. EVAL_NRT - Execute a simulation in Non-Real Time

This configuration demonstrates how to use a NON-REAL TIME (NRT) configuration.

It demonstrates the usage of the OFFLINESCHEDULER module.

The OFFLINESCHEDULER is used to precisely control the execution of modules in a non-real-time execution.

The OFFLINESCHEDULER executes the SCANeR™ programs in a precise order, defined with priority order. Moreover, OFFLINESCHEDULER gives others module precise time steps depending on their frequency.

This allows to ensure the repeatability of a scenario.

EVAL_NRT specificities:

- Realtime is disabled (OFFLINESCHEDULER module schedules the simulation):

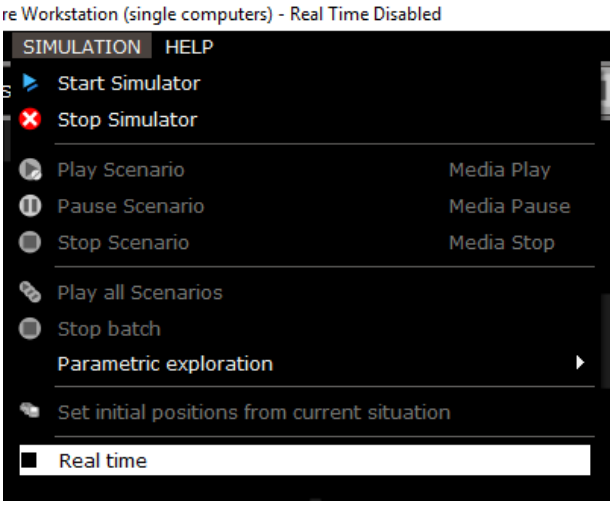

Figure 1: Real Tine Disabled

- OFFLINESCHEDULER configuration:

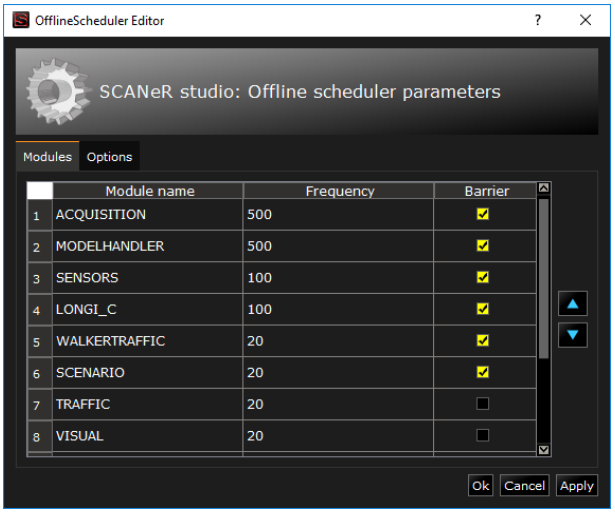

Figure 2: OFFLINESCHEDULER settings

SIMULATION TIME: SCHEDULED BY THE OFFLINESCHEDULER (the step period corresponds to the lowest denominator for all periods)

T0 (0)	T1 (0,002)	T2 (0,004)	T3 (0,006)	T4 (0,008)	T5 (0,01)	TN
ACQUISTION: STEP	ACQUISTION: STEP	ACQUISTION: STEP	ACQUISTION: STEP	ACQUISTION: STEP	ACQUISTION: STEP	Etc.
MODELHANDLER: STEP	MODELHANDLER: STEP	MODELHANDLER: STEP	MODELHANDLER: STEP	MODELHANDLER: STEP	MODELHANDLER: STEP	Etc.
SENSORS: STEP	SENSORS: WAIT	SENSORS: WAIT	SENSORS: WAIT	SENSORS: WAIT	SENSORS: STEP	Etc.
LONGI_C: STEP	LONGI_C: WAIT	LONGI_C: WAIT	LONGI_C: WAIT	LONGI_C: WAIT	LONGI_C: STEP	Etc.
WALKERTRAFFIC: STEP	WALKERTRAFFIC: WAIT	WALKERTRAFFIC: WAIT	WALKERTRAFFIC: WAIT	WALKERTRAFFIC: WAIT	WALKERTRAFFIC: WAIT	Etc.
SCENARIO: STEP	SCENARIO: WAIT	SCENARIO: WAIT	SCENARIO: WAIT	SCENARIO: WAIT	SCENARIO: WAIT	Etc.

Figure 3: OFFLINESCHEDULER management

The scenario to use with this configuration is "**EVAL_NRT**".

In this scenario, the EGO vehicle is controlled by a virtual driver (driver commands emulator).

The EGO vehicle goes straight, it is equipped with a radar sensor, when it comes closer of the pedestrian the pedestrian will cross the road without paying attention to the EGO vehicle, using the SCANeR™ API a basic ADAS system will force the EGO vehicle to brake when the pedestrian is lesser than 20 meters into the radar sensor referential system attached to the EGO vehicle.

The source code of the basic ADAS system developed with the SCANeR™ API[i] is delivered into the evaluation data package under "…\APIs\samples\EVALUATION_APIs\scanerAPI\EVAL_LONGITUDINAL_CTRL\"

- C/C++: eval_longitudinal_ctrl_pedalPos.cpp.
- Python: eval_longitudinal_ctrl_pedalPos.py.
- Simulink: eval_longitudinal_ctrl_pedalPos.slx.EVAL_TILES - Create a terrain using tiles

The tile system of the terrain mode allows users to create terrain template for quick terrain creation. In addition of this system a change localization system is available. This system allows users to dynamically switch terrain signs objects and markings dynamically depending of the localization into the world.

SCANeR™ Tiles Evaluation Package delivery:

- terrain\3DObjects\signal\US:

  US 3D objects files which can be used into the delivered tiles on a change localization.
- terrain\Tiles:

  Source files of the tiles used into the attached tutorial.
- terrain\:

  FR_Network.rnd: Terrain which uses the default French objects library.

  US_Network.rnd: Terrain which uses the default USA objects library.
- scenario\:

  FR_Scenario.sce: Scenario which uses the FR_Network.rnd.

  US_Scenario.sce: Scenario which uses the US_Network.rnd.

A tutorial is provided with the Evaluation package under "…\doc\Tutorials**A01850_Tutorial_Europe2_Tiles_EN.PDF**". It describes step by step how to use the tile system and the change localization option.

5. EVAL_DATAEXCHANGE - Exchange data

To exchange data between SCANeR™studio and an external architecture (OS, software, etc.) several solutions exist:

A. Use the SCANeR™ API.

B. Or Use the RTGateway module.

This configuration demonstrates how to use the solution B) Use the RTGateway module.

The RTGateway module is an official SCANeR™studio module made to exchange data (send/receive) between SCANeR™studio and an external program running on a Real-time or Non-Real-time platform. Communication protocols used for data exchange can be (depending of your architecture): UDP or SISCI (tested with Real-Time System iHawk).

The workflow is:

1. From Studio GUI, select the list of data to externalize and the protocol to use (UDP or SISCI). When running, the RTGateway module will automatically send selected data using the selected protocol.
2. Create an application to receive the data emitted from Studio by RTGateway.

   Optional: Via the application, send back data to Studio, RTGateway will automatically receive it and map it on export channels, so data can also be used in Studio and influence the simulation.

The scenario to use with this configuration is "**EVAL_DATAEXCHANGE**".

In this scenario, there are 2 vehicles controlled by traffic drivers.

The vehicle id 0 is equipped with a GPS sensor.

The RTGateway's filter UDP_DATAEXCHANGE:

- Sends:
  - Vehicles' Position and Speed.
  - Vehicle id 0 coordinates.
- Receives:
  - On export channel 2000: 0.01
  - On export channel 2001: 0.02
  - On export channel 2002: 0.03

To run the simulation, launch the following modules: GPSSENSOR, RT_GATEWAY, SCENARIO, TRAFFIC.

Run the following standalone program: …\AVSimulation\SCANeRstudio_2021\APIs\bin\x64\vs2013**RTGatewayIO.exe**

The position, speed and coordinates received by the program RTGatewayIO will be displayed into the GUI of it.

The export channels values are displayed into Studio APPLICATION LOG (Filter: SCENARIO / All).

6. EVAL_HEADLIGHTS - Headlights

For Headlight development and research, the following Studio modules have to be used: **NIGHTTESTMANAGER** and **AFSMANAGER**.

The **NIGHTTESTMANAGER** is used to create, manage, study and compare in real time vehicles lights. The **VISUAL** module shows the manipulation effects.

The **AFSMANAGER** is a tool of **NIGHTTESTMANAGER** used to "run" **A**daptive **F**ront light **S**ystem (AFS) Strategies.

For strategies development the **AFS API** is delivered with Studio SDK.

The aim of the **AFS API** is to allow customers to create their own AFS strategy to use in headlight simulation. The **AFS API** is available with two interfaces: Simulink and C++/DLL (legacy).

The configuration **EVAL_HEADLIGHTS** demonstrate how to develop and use an AFS strategy (ADAS) using SCANeR™studio simulation engine.

The workflow of the delivered scenario sample (**EVAL_HEADLIGHTS**) is:

- An EGO vehicle is equipped with 1 camera sensor
- An headlights configuration (set of LEDs on each beam, each LED is equipped with an AFS strategy) is applied to the EGO vehicle.
- The AFS strategy is running into a Simulink model, it reads, in Cosimulation, the camera sensor outputs, if a surrounding vehicle is into the camera sensor beam and enlightened by LED then the strategy forces the relative LED(s) to be turned off in order to do not blind the vehicle in front.

A tutorial is provided with the Evaluation package under "…\doc\Tutorials**Tutorial_Eval_Headlights _EN.PDF**". It describes step by step how to use and run the above configruation.

Watch the video: Result of the headlights configuration

7. SDK

1.1 Software

- Microsoft Visual Studio C++ 2013
- Simulink R2013/R2016
- Python 2.7

1.2 C++ source code

A Visual Studio solution is available under APIs/samples/**evaluation.sln**. It contains all the C++ source used in the evaluation data pack.

- SCANeR API
    - eval_adas
    - eval_lateral_ctrl
    - eval_longitudinal_ctrl
- RTGatewayIO

[i] The SCANeR™ API allows SCANeR™ users to develop their own program in order to communicate with SCANeR™studio. The SCANeR™ API is available in C/C++, C#, Simulink, LabView, Python, RTMaps and FMI.