



SOLUTIONS FOR VIRTUAL TEST DRIVING

CarMaker ROS Interface

User's Guide

Date: October 25, 2024
Author: David Bernhard



Table of Contents

1	Overview	4
	1.1 System Requirements.....	4
	1.2 Concept.....	5
2	Quick Start	7
	2.1 Installation and Preparation	7
	2.1.1 ROS.....	7
	2.1.2 CarMaker	8
	2.1.3 CarMaker ROS Interface	8
	2.2 Build the Example.....	9
	2.3 Run the Example	10
	2.4 Check the Communication with ROS Tools.....	10
	2.5 Parameter Manipulation	11
3	Examples	11
	3.1 HelloCM	11
	3.1.1 Topic Based Synchronization	12
4	Documentation	14
	4.1 Folder/File Overview	14
	4.2 CarMaker GUI Extension.....	16
	4.3 Infofile Parameterization.....	18
	4.3.1 General Configuration.....	18
	4.3.2 Launchfile and Rqt	19
	4.3.3 CMNode Internal and Clock Server.....	20
	4.4 Build Process.....	21
	4.4.1 ROS Workspace	21
	4.4.2 CarMaker ROS Node Shared Library	21
	4.4.3 CarMaker Executable with CarMaker C++ Interface Loader	22
	4.5 Interaction of CarMaker and the CarMaker ROS Node Shared Library	22
	4.6 CarMaker Job Scheduler	22
	4.7 Process Synchronization	23
5	Release History	24
	5.1 Version 1.1.0.....	24
	5.2 Version 1.0.0	24
	5.3 Version 0.7.0	25
	5.4 Version 0.6.8.....	25



1 Overview

“ROS (**R**obot **O**perating **S**ystem) is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.” (<https://wiki.ros.org/ROS/Introduction>)

The focus of ROS is on robotics but the field of application has become much wider. Especially for scientists and developers of Advanced Driver Assistance Systems the software framework became very interesting.

IPG Automotive is currently developing an interface and workflow to allow the usage of ROS in combination with Car-/TruckMaker. The topics code building, message passing and synchronization as well as test automation and parameterization are touched and will be explained in this document.

The functionality provided by ROS and the CarMaker Toolchain are quite wide, so there is not only one way to combine the two frameworks. This document describes an early proof-of-concept solution where the workflow of the ROS and CarMaker worlds are considered both. In the future there might be additional variations with tendency to the preferred workflow.

This document is written for users that already have deeper knowledge with the CarMaker programming interface (see CarMaker Programmer’s Guide) and general ROS usage (see e.g. <https://docs.ros.org/en/humble/Tutorials.html>)

1.1 System Requirements

General system requirements are linked to the requirements of CarMaker (see CarMaker Release Notes) and ROS (see <https://docs.ros.org/>). Table 1 shows the compatibility and existing examples for different operating systems and ROS Versions.

Platform	ROS	ROS 2
Linux 64bit	Example HelloCM with ROS workspace Tested for	Example HelloCM with ROS workspace Tested for
	<ul style="list-style-type: none"> Melodic Morenia Ubuntu 18.04 Noetic Ninjemys Ubuntu 20.04 	<ul style="list-style-type: none"> Foxy Fitzroy Ubuntu 20.04 Humble Hawksbill Ubuntu 22.04
Windows	Currently no investigation on these platforms	
Xenomai		

Table 1: Compatibility for different operating systems and ROS versions

1.2 Concept

ROS comes with a variety of functionality where the communication via topics within the same or across systems and the control of nodes via services and parameters are some of the most interesting components for modular systems exchanging data for complex control tasks. The CarMaker ROS Interface (CMRosIF) tries to support these functionalities as much as possible to allow a seamless software integration in different development states. Therefore a ROS Node has been implemented directly into the CarMaker executable (Figure 1).

CarMaker ROS Interface and ROS Node dependencies (Single Node Configuration)

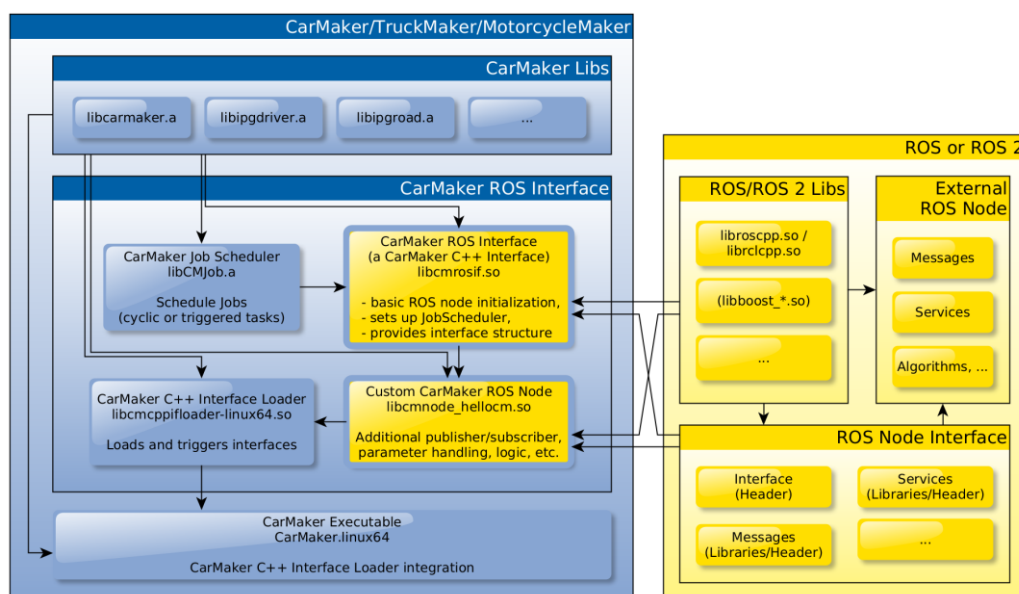


Figure 1: CarMaker ROS Interface and ROS Node Dependencies

When talking about the CarMaker ROS Interface the functionality of two main modules described below are addressed.

In order to keep the CarMaker functionalities over a wide range of application variants, the “CarMaker ROS Node” (CMNode) is not a typical ROS style standalone executable, but wrapped into a **shared library dependent on several ROS libraries** (generally ROS basic libraries and user defined libraries/packages e.g. with messages) and may have dependencies on CarMaker libraries (e.g. libcarmaker.a for access to the Vehicle struct). The C++ code for this shared library is editable by the user and can be dynamically loaded into a prepared CarMaker executable with the CarMaker C++ Interface Loader extension (might be provided as static or shared library).

The **CarMaker C++ Interface Loader** extension provides an API to the user accessible C code modules and Infile mechanism of CarMaker. The extension manages the CarMaker C++ Interface shared libraries and allows for basic parameterization via CarMaker Infile. It provides functions for common CarMaker hook points (to be called in e.g. User.c or User.cpp). While these are predefined functions and some of them are optional it is also possible to create custom functions in the CarMaker C++ Interface (e.g. CarMaker ROS Node) and load these symbols with a general mechanism assigning them to function pointers and calling them at any point in the CarMaker cycle (after basic initialization has finished).

The CarMaker ROS Node can act as a clock server by publishing the `/clock` topic with the current simulation time to other ROS Nodes (<https://wiki.ros.org/Clock>). Therefore the `/use_sim_time` parameter needs to be set to “true” before other nodes are initialized (e.g. via launch file). Figure 2 shows the rqt Node Graph for a simple Node configuration where `cm_node` publishes the `/clock` topic that is subscribed by all currently running nodes.

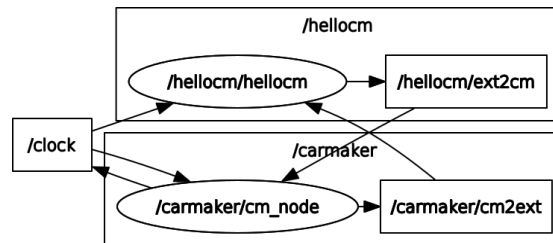


Figure 2: ROS rqt Node Graph for the HelloCM Example

CarMaker differs between the start of the CarMaker executable and a single simulation (running a TestRun). This allows to load the CarMaker ROS Node shared library at CarMaker startup or before the next simulation starts. The latter version is interesting when “restarting” the CarMaker ROS Node without a restart of the complete CarMaker executable (e.g. systems with complex initialization phase). **Currently the shared library is loaded at CarMaker startup** while the “dynamic start” of the CarMaker ROS Node is planned for the future.

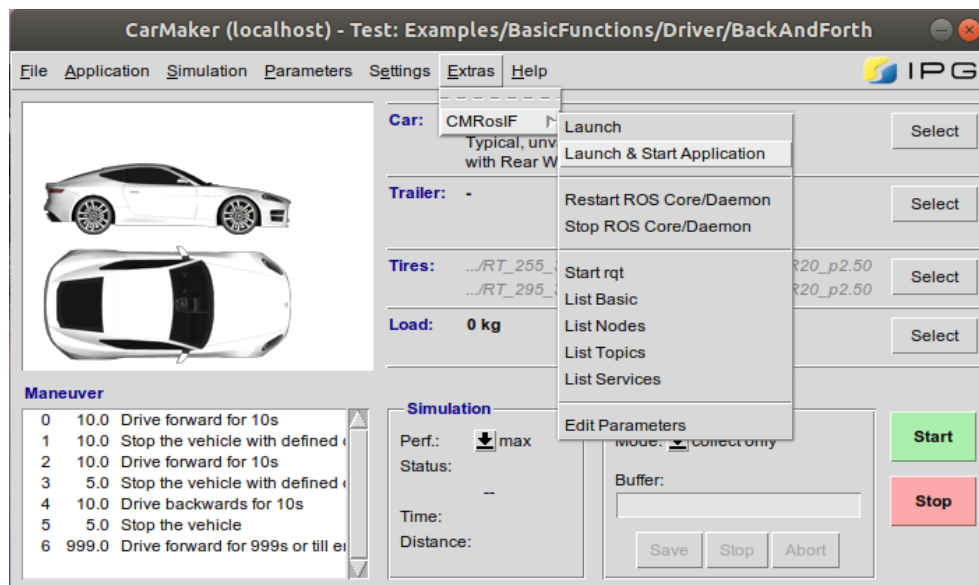


Figure 3: Overview – CarMaker GUI Extension

Finally the CarMaker ROS Interface comes with a CarMaker GUI Extension providing some basic functionalities for working with the ROS environment like editing the CMRosIF parameter file (CarMaker Infolfile) or starting default ROS programs via CarMaker GUI. More information see “4.2 CarMaker GUI Extension”.

2 Quick Start

This chapter gives a short instruction to run an example including the CarMaker ROS Interface, a ROS workspace with several packages and additional files for a specific CarMaker version.

Several scripts are not CM default and experimental for fast ramp up of this example!

2.1 Installation and Preparation

2.1.1 ROS

Follow the installation instructions for:

- ROS: <https://wiki.ros.org/ROS/Installation>
- ROS 2: e.g. <https://docs.ros.org/en/foxy/Installation/Linux-Install-Debian.html>
- Colcon: e.g. <https://docs.ros.org/en/foxy/Tutorials/Colcon-Tutorial.html#install-colcon>

By default ROS installations are located under `/opt/ros/`. For convenience you may set a symbolic link to your preferred ROS and ROS 2 distribution, so you won't have to provide the distro to your ROS workspace each time before rebuilding, e.g.:

```
1: cd /opt/ros
2: sudo ln -sfn noetic ros1
3: sudo ln -sfn foxy ros2
```

Check your ROS installations:

ROS 1:

Open a new terminal and start roscore:

```
1: source /opt/ros/ros1/setup.bash
2: roscore
```

Open a new terminal and start the talker node:

```
1: source /opt/ros/ros1/setup.bash
2: roslaunch roscpp_tutorials talker
```

Open a new terminal and start the listener node:

```
1: source /opt/ros/ros1/setup.bash
2: roslaunch roscpp_tutorials listener
```

ROS 2:

Open a new terminal and start the talker node:

```
1: source /opt/ros/ros2/setup.bash
2: ros2 run examples_rclcpp_minimal_publisher publisher_member_function
```

Open a new terminal and start the listener node:

```
1: source /opt/ros/ros2/setup.bash
2: ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
```

2.1.2 CarMaker

1. Install CarMaker according to the "InstallationGuide.pdf".
2. Configure your `~/.bashrc` for easier use, e.g. add the CarMaker installation to your path:

```
1: # Additional paths for CM
2: addpath () { for d in "$@"; do PATH="$PATH:$d"; done; }
3: addpath /opt/ipg/bin /opt/ipg/carmaker/linux64/bin /opt/ipg/carmaker/linux64/GUI
```

2.1.3 CarMaker ROS Interface

The CarMaker ROS Interface comes with a stripped down CarMaker project, so that it can be easily integrated into fresh and existing CarMaker projects, independent of the chosen CarMaker version.

Integration into a fresh CarMaker project

- First create a new CarMaker project:
 1. Start the CarMaker GUI
 2. Choose **File -> Project Folder -> Create Project...**
 3. Select the CMRosIF folder (grandparent folder of this README)
 4. Check "Sources / Build Environment"
 5. Click "Ok" and "Continue"
- Now copy the contents of `src_cmrosif/CM-<CM Major Version>/*` over to your projects source folder, e.g.:

```
1: cp src_cmrosif/CM-13/User.c src_cmrosif/CM-13/Makefile src/
```
- The source folder for CarMaker is called `src` while TruckMaker and MotorcycleMaker have an additional suffix (`_tm` and `_mm` respectively).
- If there is no subfolder for your CarMaker major version under `src_cmrosif` please use the version closest to your desired one and merge the changes manually. A visual diff viewer such as `meld` or the VS Code feature is recommended for this task:

```
1: code --diff src_cmrosif/CM-13/User.c src/User.c
2: code --diff src_cmrosif/CM-13/Makefile src/Makefile
```


Integration into existing CarMaker project

- It is recommended to version control (or backup) your existing project directory
- Make sure your current CarMaker project has a `src` folder (or the equivalent for TruckMaker/MotorcycleMaker):
 1. Start the CarMaker GUI
 2. Open your existing project using **File -> Project Folder -> Select...**
 3. Choose **File -> Project Folder -> Update Project...**
 4. Check "Sources / Build Environment"
 5. Click "Ok" and "Ok" again
- Now merge the contents of `src_cmrosif/CM-<CM Major Version>/*` over to your projects source folder, e.g.:

```
1: code --diff src_cmrosif/CM-13/User.c src/User.c
2: code --diff src_cmrosif/CM-13/Makefile src/Makefile
```
- The source folder for CarMaker is called `src` while TruckMaker and MotorcycleMaker have an additional suffix (`_tm` and `_mm` respectively).
- A visual diff viewer such as meld or the VS Code feature is recommended for this task. If there is no subfolder for your CarMaker major version under `src_cmrosif` please use the version closest to your desired one.

2.2 Build the Example

Build ROS workspace (CarMaker ROS Node shared library, messages, external nodes) and the CarMaker executable:

- For an initial build (or rebuild) of the ROS workspace it is recommended to build via CM Makefile, as it will automatically set the according CarMaker version and include paths in the CMake cache of the ROS workspace
- Open a terminal and navigate to your CarMaker projects source directory
- To build the CarMaker executable and the respective ROS workspace, run

```
1: cd <CMProjDir>/<source directory>
2: make cm-ros1 <or> make cm-ros2
```
- Subsequent builds of the ROS workspaces can be done directly without the help of the CM Makefile using the ROS build tools

```
1: cd <CMProjDir>/ros/ros_<1 or 2>ws
2: catkin_make <or> colcon
```
- If you also want to rebuild the workspace without the help of the CM Makefile or if you are sharing your workspace with other people you should define the expected CarMaker version in the appropriate CMakeLists.txt and toggle the ability to set the CM version from outside. For e.g. CarMaker 13.1.1 you would set the following in
 1. `<CMProjDir>/ros/ros1_ws/src/cmnode_hellocm/CMakeLists.txt` and
 2. `<CMProjDir>/ros/ros2_ws/src/cmnode_hellocm/CMakeLists.txt`

```
1: set(CARMAKER_VER 13.1.1 CACHE STRING "Desired CarMaker version, e.g. 11.0.1")
```

2.3 Run the Example

- Once everything is built, you can start the `./CMStart.sh` script in your project directory
 1. This script gets initially generated by the CM Makefile and may be customized to your liking
 2. It takes a single digit (e.g., 1 or 2) as an optional argument to source your ROS or ROS 2 workspace before starting CarMaker
 3. When no argument is provided, the ROS workspace that you have built first using the CM Makefile is chosen by default
- CarMaker Main GUI with menu **"Extras -> CMRosIF"** should be visible
 1. The starting order for `roscore` and nodes is important!
 2. In this version the external node is started via launch file (`roscore` is started automatically)
 3. Ensure the correct CarMaker executable has been selected via **"CM Main GUI -> Application -> Configuration/Status"**
 - "Folder"-Button : Choose `bin/CarMaker.linux64` OR `src/CarMaker.linux64` which was built previously
 4. Ensure that the correct parameters are set in the `CMRosIFParameters` file: **"CM Main GUI -> Extras -> CMRosIF -> Edit Parameters"**
 - For ROS 2, at least `Cfg.Args` and `Launch.Args` have to be changed for the example to work
 - For more information have a look at cp. 4.3 Infofile Parameterization
 5. Start external ROS Node and CarMaker Application via **"CM Main GUI -> Extras -> CMRosIF -> Launch & Start Application"**
 - If the external node was just started via "Launch" you can start the CarMaker executable via **"CM Main GUI -> Application -> Start & Connect"**
 - If you are using a ROS 2 Distribution older than Foxy, you will need to rename the fields `namespace`, `name` and `executable` inside the launch file `ros/ros2_ws/src/hellocm/launch/hellocm.launch.py` to `node_namespace`, `node_name` and `node_executable` respectfully!
 6. The CM Main GUI should show that CarMaker is running in idle state
 7. Check log messages from executable via **"CM Main GUI -> Simulation -> Session Log"**
 8. The Linux terminal should show the log messages for the external node
- Open a TestRun e.g. `Examples/Powertrain/PowertrainControl/AdaptiveCruiseControl` via **"CM Main GUI -> File -> Open..."**
- Push the "Start"-Button in CM Main GUI and check the output in the Session Log and terminal

2.4 Check the Communication with ROS Tools

- Rqt can be started conveniently via **"CM Main GUI -> Extras -> CMRosIF -> Start rqt"**
- Alternatively open a new terminal, source your ROS workspace and start `rqt`
 1. ROS: `source <CMProjDir>/ros/ros1_ws/devel/setup.bash && rqt`
 2. ROS 2: `source <CMProjDir>/ros/ros2_ws/install/setup.bash && rqt`

- Rqt lets you use several provided plugins, like
 1. running Nodes via **"Plugins -> Introspection -> Process Monitor"**
 2. node interaction via **"Plugins -> Introspection -> Node Graph"**
 3. logging via **"Plugins -> Logging -> Console"**
 4. and much more ...

2.5 Parameter Manipulation

ROS parameters might be manipulated in several ways:

- Using ROS tools

ROS

rosparam in new terminal

```
1: cd <CMProjDir>
2: source ros/ros1_ws/devel/setup.bash
3: rosparam list
4: rosparam get /hellocm/cycletime
5: rosparam set /hellocm/cycletime 50
```

ROS 2

ros2 param in new terminal

```
1: cd <CMProjDir>
2: source ros/ros2_ws/install/setup.bash
3: ros2 param list
4: ros2 param get /hellocm cycletime
5: ros2 param set /hellocm cycletime 50
```

- Using CarMaker (e.g. for test automation)
 1. Currently no direct support
 2. Tcl's `exec` command may be used with ScriptControl

3 Examples

3.1 HelloCM

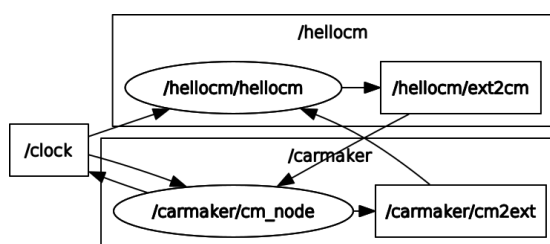


Figure 4: ROS rqt Node Graph for the HelloCM Example

- Simple example for demonstration of communication
- The current example can be used with or without hard synchronization between the CarMaker ROS Node and the external ROS Node
 1. The synchronization can be enabled/disabled in the CMRosIF Parameter file via **"CM Main GUI -> Extras -> CMRosIF -> Edit Parameters"** and the Parameter `Node.Sync.Mode`
- CarMaker can act as a clock server and provide simulation time to other ROS nodes
- Related ROS packages with source code, launch files, etc. are located in `<CMProjDir>/ros/<workspace>/src`

3.1.1 Topic Based Synchronization

The general synchronization method is described in chapter “4.7 Process Synchronization”.

The HelloCM example allows an activation of the synchronization using the parameter `Node.Sync.Mode` (see chapter “4.3 Infile Parameterization”) and provides the User Accessible Quantity `CMRosIF.HelloCM.SynthDelay` to create an artificial delay in the external ROS Node.

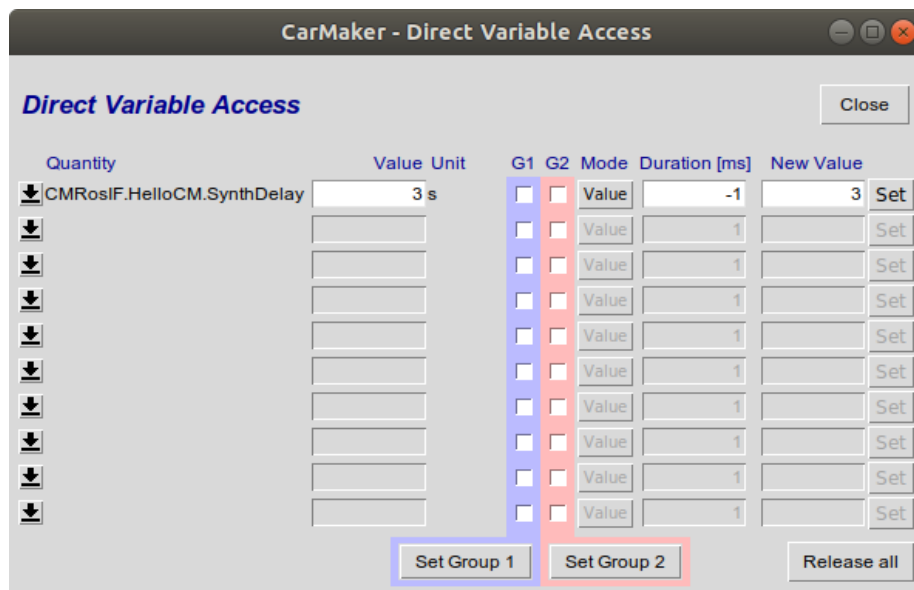
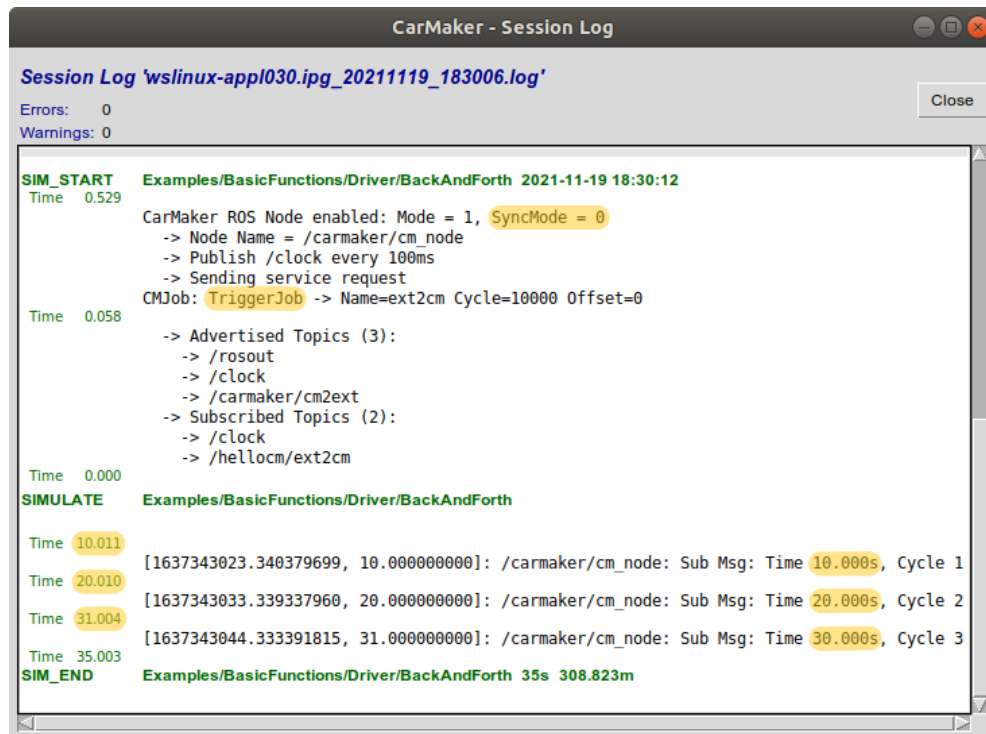


Figure 5: User Accessible Quantity to test synchronization

By default the synchronization is off and results in a behavior shown in Figure 6. The CarMaker ROS Node acts as a clock server and publishes simulation time to the ROS network. The external node reacts on the current ROS time by publishing a message with its currently known simulation time (2). This message is received by the CarMaker ROS Node at simulation time (1) with a non-constant delay. The third transmission at 30s was additionally delayed with an artificial delay of one second.



```

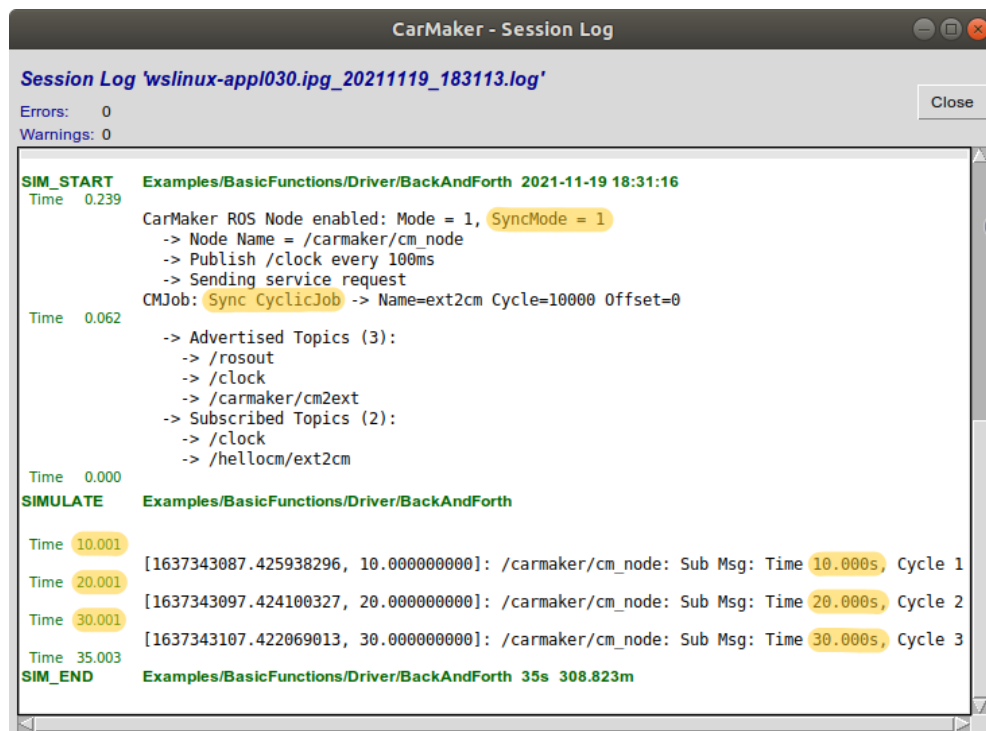
Session Log 'wslinux-appl030.ipg_20211119_183006.log'
Errors: 0
Warnings: 0

SIM_START Examples/BasicFunctions/Driver/BackAndForth 2021-11-19 18:30:12
Time 0.529 CarMaker ROS Node enabled: Mode = 1, SyncMode = 0
-> Node Name = /carmaker/cm_node
-> Publish /clock every 100ms
-> Sending service request
CMJob: TriggerJob -> Name=ext2cm Cycle=10000 Offset=0
Time 0.058
-> Advertised Topics (3):
-> /rosout
-> /clock
-> /carmaker/cm2ext
-> Subscribed Topics (2):
-> /clock
-> /hellocm/ext2cm
Time 0.000
SIMULATE Examples/BasicFunctions/Driver/BackAndForth
Time 10.011 [1637343023.340379699, 10.000000000]: /carmaker/cm_node: Sub Msg: Time 10.000s, Cycle 1
Time 20.010 [1637343033.339337960, 20.000000000]: /carmaker/cm_node: Sub Msg: Time 20.000s, Cycle 2
Time 31.004 [1637343044.333391815, 31.000000000]: /carmaker/cm_node: Sub Msg: Time 30.000s, Cycle 3
Time 35.003
SIM_END Examples/BasicFunctions/Driver/BackAndForth 35s 308.823m

```

Figure 6: Simulation without synchronization

The same simulation was started with `Node.Sync.Mode = 1` (Figure 7). Here all messages arrive deterministically after the external Node has been triggered (the Log output has a constant delay of 1ms).



```

Session Log 'wslinux-appl030.ipg_20211119_183113.log'
Errors: 0
Warnings: 0

SIM_START Examples/BasicFunctions/Driver/BackAndForth 2021-11-19 18:31:16
Time 0.239 CarMaker ROS Node enabled: Mode = 1, SyncMode = 1
-> Node Name = /carmaker/cm_node
-> Publish /clock every 100ms
-> Sending service request
CMJob: Sync CyclicJob -> Name=ext2cm Cycle=10000 Offset=0
Time 0.062
-> Advertised Topics (3):
-> /rosout
-> /clock
-> /carmaker/cm2ext
-> Subscribed Topics (2):
-> /clock
-> /hellocm/ext2cm
Time 0.000
SIMULATE Examples/BasicFunctions/Driver/BackAndForth
Time 10.001 [1637343087.425938296, 10.000000000]: /carmaker/cm_node: Sub Msg: Time 10.000s, Cycle 1
Time 20.001 [1637343097.424100327, 20.000000000]: /carmaker/cm_node: Sub Msg: Time 20.000s, Cycle 2
Time 30.001 [1637343107.422069013, 30.000000000]: /carmaker/cm_node: Sub Msg: Time 30.000s, Cycle 3
Time 35.003
SIM_END Examples/BasicFunctions/Driver/BackAndForth 35s 308.823m

```

Figure 7: Simulation with topic based synchronization

4 Documentation

4.1 Folder/File Overview

The paths below are relative to the CarMaker Project Directory and describe the structure and contents after the integration of the CMRosIF files. Files and folders in *italic* will be available after building. The structure can be adapted for different use cases (references in files will have to be updated).

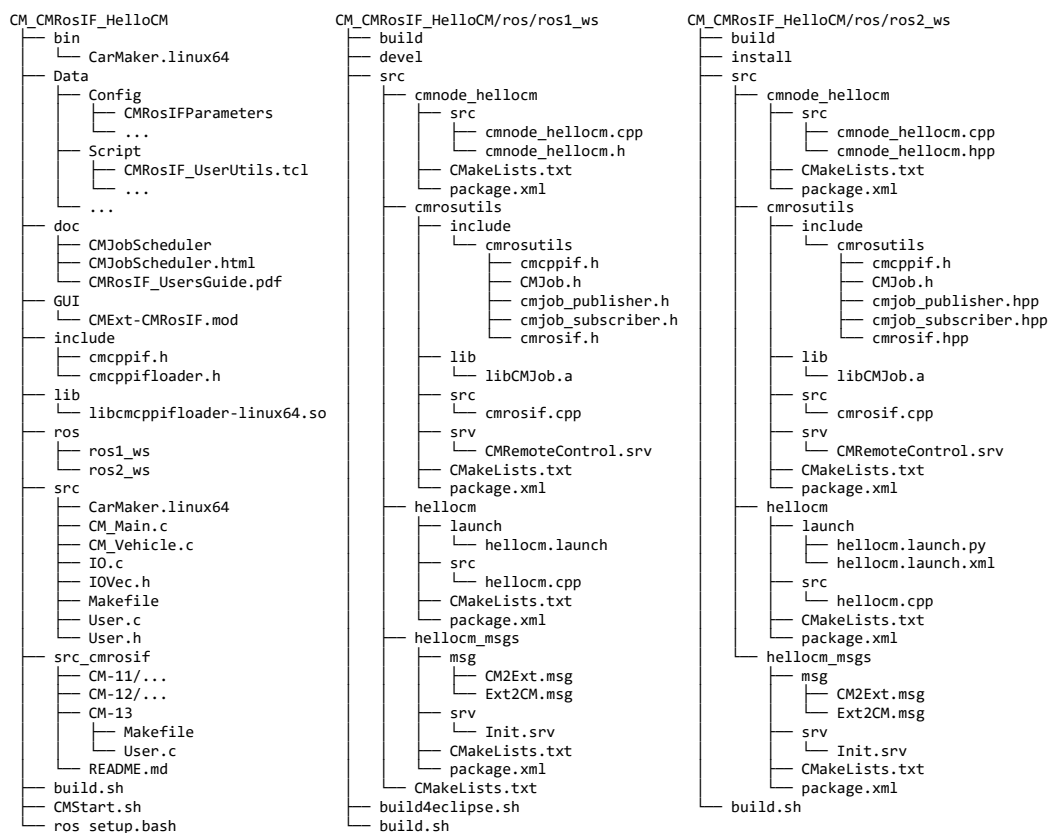


Figure 8: CM project folder overview with provided ROS and ROS 2 workspace examples

Description of most relevant folders/files:

bin/

- Location of the CarMaker executable after installing (make install)

Data/Config/CMRosIFParameters

- CarMaker Infofile with parameters for the CMRosIF and CarMaker ROS Node
 - Enable/disable interface/clock server
 - CarMaker ROS Node remapping arguments,
 - i.e. `Cfg.Args = __ns:=MyNamespace` to push down all topics, ...

- Accessible via "**CM Main GUI -> Extras -> CMRosIF -> Edit Parameters**"
- More Information see "4.3 Infofile Parameterization"

doc/

- Documentation for additional modules, like the CMRosIF and CMJobScheduler

GUI/

- Folder for CarMaker GUI extension of current CarMaker project directory
- The mod files can be copied to the CarMaker installation folder to be available globally
 1. CMExt-CMRosIF.mod: see chapter "4.2 CarMaker GUI Extension"

include/

- Folder with additional include files
- The headers `cmcppif.h` and `cmcppifloader.h` for CarMaker C++ Interfaces are located in this folder

lib/

- Additionally linked libraries of the CarMaker executable like the CarMaker C++ Interface Loader library `libcmcppifloader-linux64.so`

src/

- CarMaker source folder for building the CarMaker executable
- Needs to be updated once using the files located under `src_cmrosif/`
- Makefile
 1. Default CarMaker Makefile with additional flags/libraries for using the CarMaker C++ Interface Loader module and additional goals to build the ROS workspaces
 2. Building CarMaker executable with CarMaker C++ Interface Loader and ROS workspaces
- User.c
 1. Loading and triggering CarMaker C++ Interfaces, such as the CMRosIF, using the CMCppIFLoader lib/libcmcppifloader-linux64.so
 2. see include/cmcppifloader.h for more details

src_cmrosif/CM- <CarMaker major version>

- template files (`Makefile`, `User.c`) to update the CarMaker source folder manually

build.sh

- Builds CarMaker and the ROS workspaces. Experimental for fast ramp up of the examples!
- Using the Makefile `<CMProjDir>/src/Makefile` of your projects source folder is preferred

CMStart.sh

- Sources your ROS workspace and starts the CarMaker GUI with the GUI extension. File is created during the first run of `build.sh` or `make` within the CarMaker source directory.

ros_setup.bash

- helper script for sourcing the appropriate ROS workspace
- used by CMRosIF GUI extension and CMStart.sh script

ros/ros1_ws/ and ros/ros2_ws/

- Native ROS workspaces (ROS: catkin, ROS 2: ament) with user packages (messages, nodes, ...)
- Including topics and services as well as the external ROS Node and the CarMaker ROS Node shared library for the HelloCM example
- The workspace might also be set as a symbolic link to an already existing ROS workspace or referenced in the CM Makefile

src/cmnode_hellocm

- Source code for the CarMaker ROS Node shared library
- Derives from CarMaker ROS Interface base class located under the cmrosutils package
- Created as a CarMaker C++ Interface
- Place for customer-specific changes

src/cmrosutils

- General utilities when using ROS in combination with CarMaker
- Includes base class for the general CarMaker ROS Interface
- CarMaker ROS Interface base class derives from more general CarMaker C++ Interface class to ensure compatibility with the CarMaker C++ Interface Loader

src/hellocm

- Source code for the external ROS node of the example (independent of CarMaker)

src/hellocm_msgs

- Message and Service definitions used by the nodes above

build.sh or build4eclipse.sh

- Build scripts for the ROS workspace. While build.sh runs a normal build, build4eclipse.sh prepares the ROS .build/ folder for an eclipse project that can be imported to eclipse (inside eclipse: "File -> Import -> Existing Projects into Workspace")

4.2 CarMaker GUI Extension

The CarMaker ROS Interface comes with an optional CarMaker GUI extension module. The extension provides an extra menu "**CM Main GUI -> Extras -> CMRosIF**" in the CarMaker Main GUI. The menu extension is included in the file CMExt-CMRosIF.mod. This file can be placed inside <CMInstDir>/GUI to be globally available or inside the CarMaker project directory and loaded via an additional command line argument `-ext <filename>.mod` when starting the CarMaker GUI, e.g. if the mod file is located in the folder <CMProjDir>/GUI the command could be

```
| 1: CM . -ext GUI/CMRosIF.mod or CM-13.1.1 . -ext GUI/CMRosIF.mod .
```


For the CMRosIF example, the command line is located inside the `CMStart.sh` script. It is required to add the CarMaker installation path to the systems search path (please check chapter “2.1.2 CarMaker” for additional information) to use it.

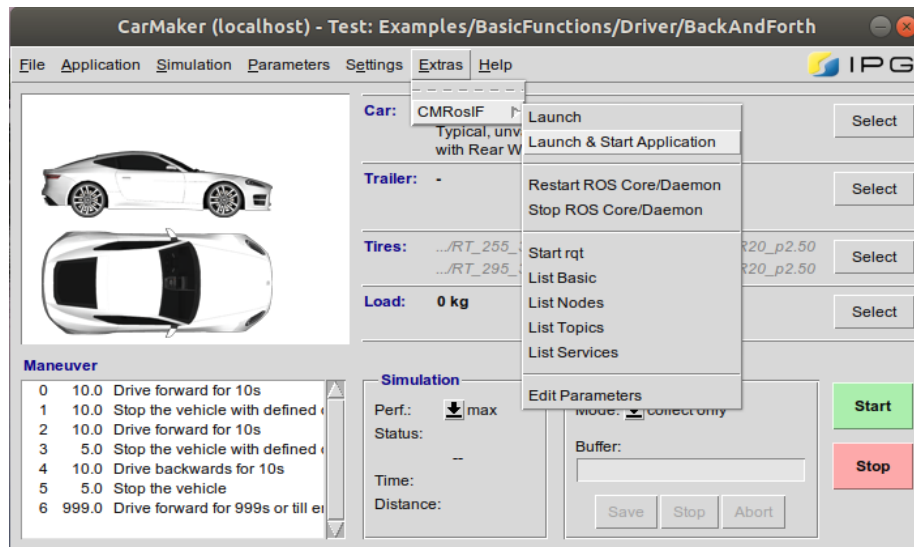


Figure 9: CarMaker GUI Extension

Following commands are currently available:

Launch

This command starts a new terminal, sources the script `<CMProjDir>/ros_setup.bash` and executes ROS Launch. If the feature `TerminalCmd` is enabled, previously started terminals by the Launch feature will be stopped first. The arguments for ROS Launch can be parameterized in the Infofile `<CMProjDir>/Data/Config/CMRosIFParameters`. (see chapter “4.3 Infofile Parameterization”)

Launch & Start Application

This command works like the command “Launch” described above, but additionally the CarMaker executable is started. An already running executable will be stopped. This allows a complete start/restart of the simulation setup. Several parameters influence the startup (e.g. `Cfg.Features`). Please check the chapter “4.3 Infofile Parameterization” for more information. Please ensure that the correct CarMaker executable is selected (“CM Main GUI -> Application -> Configuration/Status”)

<Restart/Stop> ROS Core/Daemon

“Restart” stops the currently running roscore and starts again. “Stop” just stops the process. Normally not necessary when using the command “Run launch file” where the roscore is automatically started with the other ROS nodes.

Start rqt

This command starts a new terminal, sources the script `<CMProjDir>/ros_setup.bash` and executes the ROS program `rqt`. The arguments for `rqt` can be parameterized in the Infofile `<CMProjDir>/Data/Config/CMRosIFParameters`. (see chapter “4.3 Infofile Parameterization”)

List <Basic/Nodes/Topics/Services>

These commands run the corresponding ROS programs rostopic, rosnodet, rosservice and prints the output to the CarMaker ScriptControl window.

Edit Parameters

This command opens the Infofile <CMProjDir>/Data/Config/CMRosIFParameters for editing. (see chapter " 4.3 Infofile Parameterization").

4.3 Infofile Parameterization

Following parameters can be used to parameterize the CarMaker ROS Interface. The parameters allow for a modification of the internal mechanism and e.g. the arguments provided to the CarMaker ROS Node in the initialization function `CMCppIFLoader_init()`.

The parameters are managed in different groups with their own prefix

- `Cfg.*` : General configuration of CMRosIF
- `Launch.*`: Parameters related to the command **"CM Main GUI->Extras->Run launch file"**
- `rqt.*` : Parameters related to the command **"CM Main GUI->Extras->Start rqt"**
- `Node.*` : Parameters used inside CarMaker ROS Node. Depends on node design!

4.3.1 General Configuration

Cfg.Lib.Path

This parameter is not optional!

Path to the CarMaker ROS Node shared library. The path can be absolute or relative to current CarMaker project directory or available within the PATH variable, e.g. `LD_LIBRARY_PATH`. The shared library is currently loaded at startup of the CarMaker executable.

Example: `Cfg.Lib.Path = libcmnode_hellocm.so`

Cfg.Mode

This parameter is optional! Default = 1

The CarMaker C++ Interfaces can be disabled (`Cfg.Mode = 0`). In this case the interfaces shared library is not loaded after starting the CarMaker executable and the CarMaker ROS Node will thus not be available. User functions obtained with

`CMCppIFLoader_getSymbol()` will be NULL pointers and will have to be managed by the user (e.g. `if (MyFunc != NULL) MyFunc();`)!

Cfg.Name

This parameter is optional! Default = "cm_node"

Changes the name of the CarMaker ROS Node provided to the CMRosIF GUI. To dynamically change the actual name of the node, use remapping arguments that can be set using the `Cfg.Args` infofile parameter.

Cfg.Args

This parameter is optional! Default = ""

The parameter string is mapped to the `argc` and `argv` arguments and eventually evaluated by the ROS init functions. This allows the user to provide arguments in a style of a ROS standalone node where the arguments `argc` and `argv` are provided by the `main()` function (e.g. for remapping arguments, ...).

Remapping arguments can't be provided directly as arguments to the CarMaker executable!

Example with `Cfg.Args = __ns:=/carmaker`:

`CarMakerRosInterface::init()` will provide `argc=2` and `argv[1]="__ns:=/carmaker"` to the ROS init function: Node name, Topics, etc. will be pushed down to the ROS namespace `/MyVhcl` (so the node name `/cm_node` will become `/carmaker/cm_node`).

Cfg.Features

This parameter is optional! Default = "TerminalCmd"

The parameter allows a space separated list with different special features available for the CMRosIF.

- `AutoStart` (experimental)
 1. Enables an automated mechanism when CarMaker TestRun is started
 - Stopping ros launch and CM executable
 - Start of ros launch file and CM executable
 - calls `Cfg.LaunchProc` in context `startsim`
- `TerminalCmd` (experimental, Ubuntu only)
 1. Mechanism to handle multiple Terminals
 2. Automatic termination of launch terminal
 3. Optional multi terminal/tab mechanism
 - Managing different tabs and terminals and running multiple launch files in the same (one tab for each) or multiple terminals
 - Create additional terminals with Linux commands
 - Check `Cfg.UserUtils.FPath` and `Cfg.LaunchProc`
 4. Additional parameterization via keys
 - `<pre> = Cfg.TerminalCmd`
 - `<pre>.startwait` = Time to wait in ms after start command was executed
 - `<pre>.stopwait` = Time to wait in ms after terminal stop request

4.3.2 Launchfile and Rqt

Launch.Args

Arguments provided to the ROS program `roslaunch/ros2 launch` when the GUI command **"CM Main GUI >Extras >CMRosIF->Run launch file"** is used.

Example:

`Launch.Args = hellocm hellocm.launch use_sim_time:="true"` will run the launch file `hellocm.launch` from ROS package `hellocm` providing the argument `use_sim_time:="true"`.

rqt.Args

Arguments provided to the ROS program `rqt` when the GUI command “**CM Main GUI->Extras->CMRosIF->Start rqt**” is used.

4.3.3 CMNode Internal and Clock Server

Node.Mode

Usage depends on node design! See source files of CarMaker ROS Node for more information.

The intention of this parameter is to set an node internal mode (0=Off, 1=Enabled).

Node.Sync.Mode

Usage depends on node design! See source files of CarMaker ROS Node for more information.

The synchronization mechanism is currently under investigation. Currently external ROS nodes can be synchronized only via the `/clock` Topic published by the CarMaker ROS Node. But CarMaker does not wait for the answer, so incoming Topics may arrive delayed and not in an deterministic way.

The intention of this parameter is to set a node internal synchronization mode (0=NoSync, 1=SyncViaTopics).

Node.Sync.TimeMax

Usage depends on node design! See source files of CarMaker ROS Node for more information.

Defines the timeout in seconds if synchronization is enabled. By default CarMaker will throw an error and stop the current TestRun if the expected topic is not received within this timeout.

Node.UseSimTime

Usage depends on node design! See source files of CarMaker ROS Node for more information.

If this parameter is set to 1 CarMaker will act as a ROS Clock Server by setting the ROS parameter `/use_sim_time` and publishing the current simulation time to `/clock` every `Node.nCyclesClock` cycles.

Node.nCyclesClock

Usage depends on node design! See source files of CarMaker ROS Node for more information.

The clock update interval determines how often the ROS clock is updated, with a value of 1 being every CarMaker simulation cycle (which by default is 1 millisecond).

4.4 Build Process

Chapter “2.3 Build the Example” shows how to build the example using the CarMaker Makefile. All included steps can be executed by hand as described below.

4.4.1 ROS Workspace

The ROS workspace (e.g. `<CMProjDir>/ros/ros1_ws`, native ROS workspace) contains the script `build.sh`, that makes some preparation (e.g. sourcing `setup.bash` from ROS installation) and executes the default ROS command `catkin_make` (ROS default) or `colcon build` (ROS 2 default). All packages located in the `src` folder below the workspace will be built. To ignore a package, add an empty file called `CATKIN_IGNORE` (ROS) or `AMENT_IGNORE` (ROS 2) to the root level of a package (e.g. `src/hellocm/CATKIN_IGNORE`).

The location of the ROS workspace inside a CarMaker Project Directory is optional. The ROS workspace can also be in a different place and linked via symbolic link. Without a link inside `<CMProjDir>/ros/` some features shown with the provided example might not work.

For detailed information, check <https://wiki.ros.org/catkin/workspaces> or <https://colcon.readthedocs.io/en/released/user/what-is-a-workspace.html>.

For the provided example all code with ROS dependency is located in this workspace (e.g. external ROS Node and the CarMaker ROS Node shared library, ...).

4.4.2 CarMaker ROS Node Shared Library

The CarMaker ROS Node shared library is built within the ROS workspace in its own ROS package (e.g. `<CMProjDir>/ros/ros1_ws/cmnode_hellocm`).

- The path to the CarMaker ROS Node shared library which is dynamically loaded inside the CarMaker executable can be adapted by editing the parameter `Cfg.Lib.Path` (see ch. “4.3.1 General Configuration”)
- The shared library contains the `REGISTER_CARMAKER_CPP_IF` macro in order to be loadable by the interface loader
- Additional user defined functions can be created inside the library and called from e.g. `User.c` via function pointers (see example code for `cmnode_hellocm.cpp`, `User.c` and `cmcppifloader.h`)
- CarMaker Data (e.g. Sensors and Vehicle Model) can be directly accessed inside the CarMaker ROS Node shared library. Just add the necessary headers from `<CMInstDir>/include/`. Please note that the shared library will depend on CarMaker libraries and the CarMaker version at compile time! For a new CarMaker Version (especially new major releases) the library needs to be recompiled.
- The CarMaker Version (path and version number) can be adapted in the `CMakeLists.txt`.

Please check `CMakeLists.txt` and `package.xml` inside the `cmrosutils` and `cmnode_hellocm` packages for more information.

4.4.3 CarMaker Executable with CarMaker C++ Interface Loader

The CarMaker executable with the CarMaker C++ Interface Loader is built using the CarMaker Makefile, e.g. located in `<CMPProjDir>/src`. To build it open a new terminal and execute `make install` inside this folder (more information see “Programmers Guide”). The CarMaker executable for this example usually only needs to be compiled after modifications in the CarMaker user modules (`User.c`, ...) or when the CarMaker C++ Interface API changes.

The CarMaker C++ Interface Loader currently consists of a header i.e.

`<CMPProjDir>/include/cmcppifloader.h` and a shared library i.e.

`<CMPProjDir>/lib/libcmcppifloader-linux64.so`. The shared library is linked with the CarMaker executable and is automatically loaded at startup.

4.5 Interaction of CarMaker and the CarMaker ROS Node Shared Library

The CarMaker ROS Node shared library is not directly linked to the CarMaker executable. When calling the loading function of the CarMaker C++ Interface Loader inside `User.c` the CarMaker ROS Node shared library parameterized by `Cfg.Lib.Path` (see 4.3 Infofile Parameterization) will be loaded during runtime of the CarMaker executable. Commonly used default hook points already exist (see `cmcppifloader.h` and `cmcppif.h`). These hook functions call the appropriate member functions of the CarMaker ROS Node shared library.

For example, `CMCppIFLoader_testrunStartAtBegin()` will eventually call the the member function `CMNodeHelloCM::userTestrunStartAtBegin()`.

The `CMCppIFLoader_load()` with the appropriate interface name (“CMRosIF”) has to be called before any other function from the `CMCppIFLoader` API.

Additional user functions may be defined as `extern “C”` inside the CarMaker ROS Node shared library. The function `CMCppIFLoader_getSymbol()` allows to search for functions/symbols inside the recently loaded CarMaker C++ Interface library and returns a function pointer that can be used e.g. in `User.c`. Check `User.c` and the shared library source code for an example.

Variables from the CarMaker environment can be directly accessed by including the relevant CarMaker headers (e.g. `Vehicle.h` for vehicle velocity, ...) from the CarMaker installation directory (e.g. `/opt/ipg/carmaker/linux64/include`) in the CarMaker ROS Node shared library. Libraries for linking are located in the `lib/` folder of the CarMaker installation directory.

4.6 CarMaker Job Scheduler

The job scheduler is used to execute one or more cyclic tasks (e.g. publish and subscribe messages, copy data from message buffer to vehicle model, ...) dependent on e.g. the CarMaker cycle number relative to simulation start. This allows to publish data e.g. every 10ms with an offset of e.g. 5ms.

For an example on how to set up a publisher and subscriber, please have a look at:

- `cmrosutils/cmjob_publisher.h[pp]`,
- `cmrosutils/cmjob_subscriber.h[pp]`,
- `cmnode_hellocm.h[pp]` and
- `cmnode_hellocm.cpp`.

For more details, please refer to the Doxygen documentation available at
<CMProjDir>/doc/CMJobScheduler.html.

4.7 Process Synchronization

By default ROS is a non-deterministic software framework that transfers messages between several nodes using the publish/subscribe mechanism. The actual time until a message is received depends on system workload and the transmission path. Running multiple processes independently may result in non-reproducible simulations. If running the simulation in soft real time the effect might be small and therefore acceptable for many use cases, but running CarMaker with maximum simulation speed small deviations in the message timing will have an impact on simulation results.

Therefore a simple example of a topic based synchronization between an external ROS Node and the CarMaker ROS Node is available that forces CarMaker to wait until the anticipated message is received. The synchronization is based on knowledge about the expected cycle time of the external node and uses an internal cycle counter inside the CarMaker Node.

The mechanism may be used in different ways

- Simulation time based synchronization
 1. The ROS `/use_sim_time` mechanism needs to be activated
 2. CarMaker ROS Node has to act as a clock server
 3. ROS Timer inside external ROS Node that reacts on changes in simulation time
 4. Provide cycle time and topic name for synchronization to CarMaker ROS Node
- Data triggered synchronization
 1. ROS Node interaction with one or more calculation chains, each chain finishes with a topic in the CarMaker ROS Node that is used for synchronization
 2. One ore more subscription(s) in external ROS Node(s)
 - Includes algorithm, etc.
 - ROS publish at the end of calculation chain
 3. Provide cycle time and topic name for synchronization to CarMaker ROS Node
 4. CarMaker ROS Node waits in specific cycle until all messages have arrived

The simulation time based synchronization is implemented in the example described in chapter “3.1 HelloCM”. The effect of synchronization is demonstrated in the chapter “3.1.1 Topic Based Synchronization”.

5 Release History

5.1 Version 1.1.0

General

- Add compatibility for CarMaker major versions 11, 12 & 13
- Switch default CMRosIF settings to use ROS 2
- Make CMCppIF compatible with C++ compiled CarMaker executables
- Add ability to run CarMaker with CMRosIF outside of project directory
- Simplify finding CarMaker header files for compiling the CarMaker ROS node
- Install custom CarMaker executable as symbolic link to <CMProjDir>/bin
- Increase verbosity of some log outputs (synchronization mode and disabled CM node)
- Fix `rosidl_target_interfaces()` deprecation warning in ROS2 Humble and newer

CM Job Scheduler

- Update JobScheduler to v1.2.0
- Activate sync mode by default
- Publish simulation time @1000Hz by default

5.2 Version 1.0.0

General

- Renamed CarMaker ROS Interface loading library to CarMaker C++ Interface Loader
- Defined CarMaker C++ Interface as abstract class
- CarMaker ROS Interface inherits from CarMaker C++ Interface
 1. Implements general ROS functionality
- User example CMNodeHelloCM inherits from CarMaker ROS Interface
 1. Implements user-specific functionality
- Replaced job mechanism with CMJobScheduler
- Refactored ROS workspaces
- Tested with ROS 2 Foxy Fitzroy and Galactic Geochelone

5.3 Version 0.7.0

General

- CMRosIF including the GUI distinguish between ROS and ROS 2 depending on which version is sourced at startup
- Integrated native ROS 2 workspace with HelloCM example
 1. tested with Eloquent Elusor and Dashing Diademata
- Simplified installation and build process that supersedes the need of changing CM versions in multiple places
- Fixed a deprecation warning by `gnome-terminal` in `TerminalCmd`

5.4 Version 0.6.8

General

- Update `User.c` and `Makefile` for CarMaker executable in `<CMProjDir>/src/`
- Default target for ROS build and examples is `devel` (before `install`)
- Add script `<CMProjDir>/ros/ros1_ws/src/build4eclipse.sh`
 1. Builds a catkin workspace with eclipse project files inside the `build` folder and allows a direct import as eclipse project
 2. According to “Catkin-y approach” described on <http://wiki.ros.org/IDEs#Eclipse>
 3. The script can be called manually on demand
- Now the demo package is independent from CarMaker version and can be easily integrated into an already existing CarMaker Project Directory

Examples: HelloCM

- New example for topic based synchronization
- New job mechanism for cyclic data publishing on CMNode Side
- Renamed ROS messages and restructured global variables for CMNode and external ROS Node

GUI

- New menu entry `Launch & Start Application`
- New experimental feature `TerminalCmd`
 1. Manages terminals started via CarMaker GUI e.g. menu entry **“CM Main GUI -> Extras -> CMRosIF-> Launch”**
 2. More information see chapter “4.3 Infofile Parameterization” and parameter `Cfg.Features`