CarMaker ROS Interface

Proof of Concept

Date: May 23rd, 2019
Author: IPG

Table of Content

1 Overview

“ROS (Robot Operating System) 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.” (http://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 ROS world and CarMaker world are considered both. For future there might be additional variations with tendency to the preferred workflow.

This document is written for users that already have deeper knowledge in CarMaker programming interface (see CarMaker Programmers Guide) and general ROS usage (see e.g. http://wiki.ros.org or http://wiki.ros.org/ROS/Tutorials)

1.1 System Requirements

General system requirements are linked to the requirements of CarMaker (see CarMaker ReleaseNotes) and ROS (see http://www.ros.org/). Table 1-1 shows the compatibility and existing examples for different operating systems and ROS Versions.

Platform	ROS 1	ROS 2
Linux Tested for: • Ubuntu 16.04, 64bit	Example HelloCM with ROS workspace Tested for • kinetic kame	Currently not supported. The basic CMRosIF library is independent from ROS version and can be reused. Adaption in build process, open source code for shared library with CarMaker ROS Node and ROS Workspace are necessary. Modification can be done by user. CM GUI extensions will not work (other ROS programs need to be executed).
Windows	Currently no investigation on this platforms	
Xenomai		

Table 1-1: *Compatibility for different operating systems and ROS versions*

1.2 Concept

ROS comes up with a variety of functionality where the communication via Topics on same or distant systems in the same network and control of Nodes via Services and Parameters are some of the most interesting components when thinking of 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).

Currently no support for ROS2. Please check chapter System Requirements for more information.

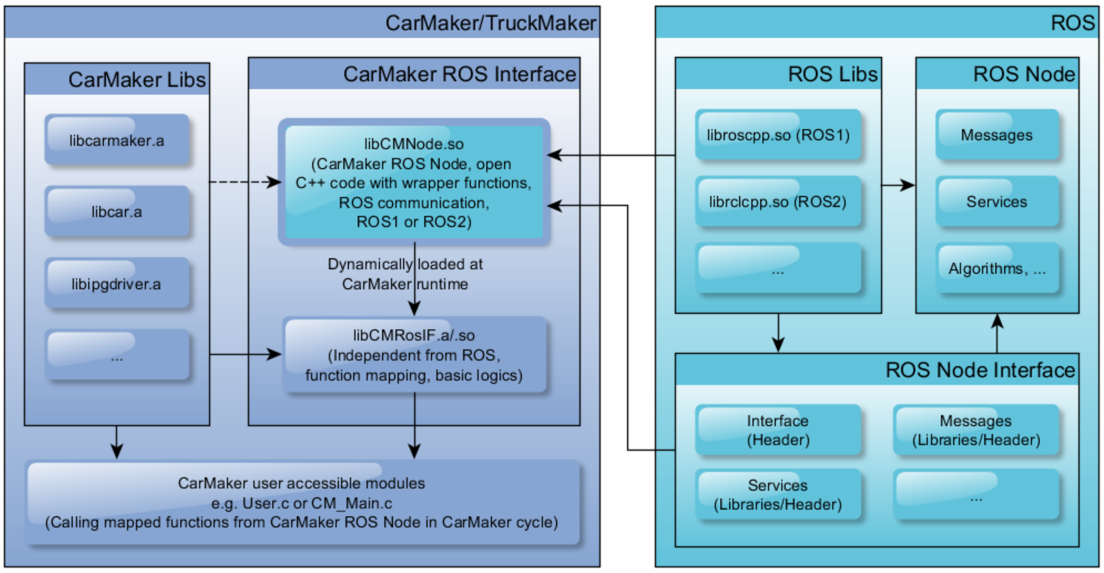

CarMaker/TruckMaker
CarMaker Libs
libcarmaker.a
libcar.a
libipgdriver.a
...
CarMaker ROS Interface
libCMNode.so (CarMaker ROS Node, open C++ code with wrapper functions, ROS communication, ROS1 or ROS2)
Dynamically loaded at CarMaker runtime
libCMRosIF.a/.so (Independent from ROS, function mapping, basic logics)
CarMaker user accessible modules e.g. User.c or CM_Main.c (Calling mapped functions from CarMaker ROS Node in CarMaker cycle)
ROS
ROS Libs
libroscpp.so (ROS1)
librclcpp.so (ROS2)
...
ROS Node
Messages
Services
Algorithms, ...
ROS Node Interface
Interface (Header)
Messages (Libraries/Header)
Services (Libraries/Header)
...

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 dependency to the 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 a CarMaker ROS Interface extension (might be provided as static or shared library).

The **ROS independent CarMaker ROS Interface** extension provides an API to the user accessible C code modules and Infofile mechanism of CarMaker. The extension manages the ROS dependent shared library above and allows a basic parameterization (e.g. Node name or remapping arguments) of the CarMaker ROS Node via a CarMaker Infofile. It provides functions for common CarMaker Hook Points (e.g. in User.c or User.cpp). While these are predefined functions and some of them are optional it is also possible to create own functions in the CarMaker ROS Node library and get these symbols with a general mechanism allowing to assign 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 the other ROS Nodes (http://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 CMNode publishes the “/clock” topic that is subscribed by all currently running nodes.

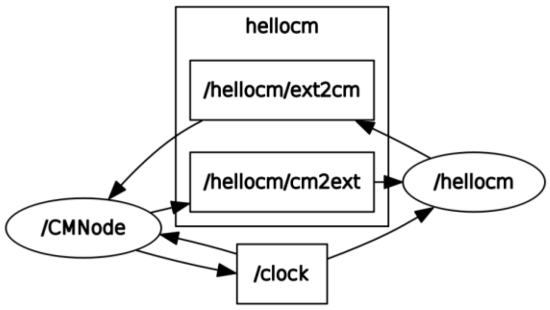


Figure 2: ROS rqt Node Graph for the HelloCM Example

CarMaker differs between the start of the CarMaker executable and a single simulation (running a TestRun). This allows to load the shared library with CarMaker ROS Node at CarMaker startup or before next simulation starts. Second 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 future.

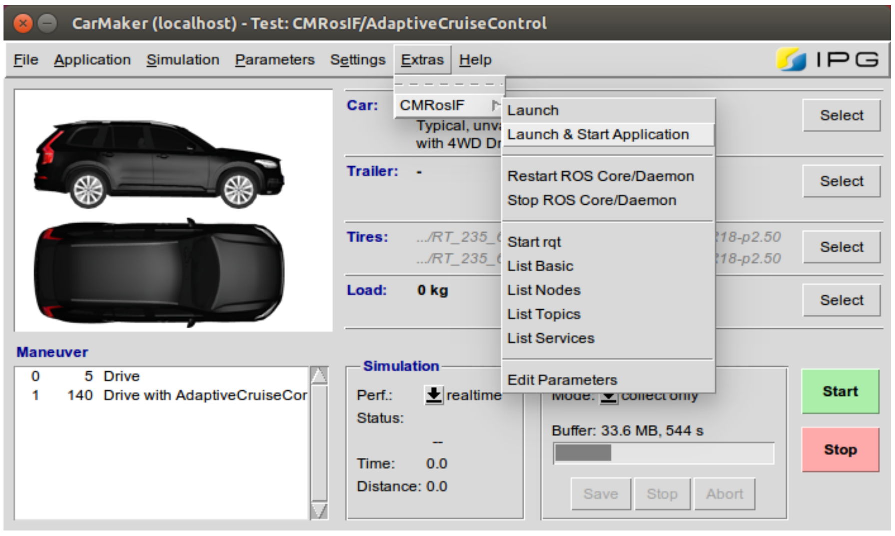


Figure 3: Overview – CarMaker GUI Extension

Finally the CarMaker ROS Interface comes up with a CarMaker GUI Extension providing some basic functionalities with the ROS environment like editing the CMRosIF parameter file (CarMaker Infofile) or wrapped default ROS programs started/handled via CarMaker GUI. More information see “4.3 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.

2.1 General Information

- Several scripts are not CM default and experimental for fast ramp up of this example!
- Current interface and build process is a prototype and will be improved in future

2.2 Installation and Preparation

2.2.1 ROS

- Follow the installation instructions on http://wiki.ros.org/ROS/Installation
- By default ROS installation is located in **"/opt/ros/"**
- Create **symbolic link "/opt/ros/ros1"** that points to e.g. "/opt/ros/kinetic"
    - e.g. “cd /opt/ros; sudo ln -sfn kinetic ros1”
    - this is only to simplify usage of different ROS versions
    - otherwise you have to change paths with every ROS update (e.g. in scripts)
- Check ROS installation
    - roscore
        - Open a new terminal
        - source /opt/ros/ros1/setup.bash
        - roscore
    - Talker
        - Open a second terminal
        - source /opt/ros/ros1/setup.bash
        - rosrun roscpp_tutorials talker
    - Listener
        - Open a third terminal
        - source /opt/ros/ros1/setup.bash
        - rosrun roscpp_tutorials listener

2.2.2 CarMaker and CMRosIF

- Install CarMaker according to "InstallationGuide.pdf"
  - There is a special installation routine for CarMaker on Ubuntu! (CM Version < 8.0)
- The package with the CarMaker ROS Interface is prepared for a specific CarMaker version. Please check chapter **"4.1 Installation"** for additional information!

2.3 Build necessary Parts

Build ROS Messages, shared library with CarMaker ROS Node and CarMaker executable:

- Open terminal and navigate to the CarMaker Project Directory
- Execute **"./build_cmrosif.sh**" to build ROS Messages, external ROS Node, shared library with CarMaker ROS Node and CarMaker executable with the CMRosIF
  - Normally CarMaker executable with CMRosIF needs to be built only once
  - All steps can be done manually. The script is only a small workaround for quick ramp up!
    - *cd ros/ros1_ws/; ./build.sh*
    - *cd ../../src; make* and optional *make install*

2.4 Run the Example

- Once everything is built you can run the script "**./CMStart.sh**"
- CarMaker Main GUI with menu "Extras -> CMRosIF" should be visible
  - The starting order for roscore and Nodes is important!
  - In this version the external node is started via launch file (roscore is started automatically)
  - 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" that was built previously
    - Depends on your workflow! Special search order is used (see "CM Main GUI -> Help -> User's Guide") e.g. you can also write "CarMaker.linux64" (first "bin/" in current project is checked, then Installation folder)
  - Start external ROS Node and CarMaker Application via "CM Main GUI -> Extras -> CMRosIF -> Launch & Start Application"
    - If external node was just started via "Launch" you can start the CarMaker executable via " CM Main GUI -> Application -> Start & Connect"
  - The CM Main GUI should show that CarMaker is running in idle state
  - Check log messages from executable via "CM Main GUI -> Simulation -> Session Log"
  - The Linux terminal should show the log messages for this 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

2.5 Check the Communication with ROS Tools

- rqt can be started as mentioned below or directly
  via "CM Main GUI -> Extras -> CMRosIF -> Start rqt"
- Open new terminal
- source <path to setup.bash in ROS user workspace>
  - e.g. “<CMProjDir>/ros/ros1_ws/devel/setup.bash”
- start “rqt” and use the provided Plugins
  - Running Nodes via "Plugins -> Introspection -> Process Monitor"
  - Nodes interaction via "Plugins -> Introspection -> Node Graph"
  - Logging via "Plugins -> Logging -> Console"
  - ...

2.6 Parameter Manipulation

- Parameters might be manipulated on several ways
- ROS
  - rosparam in terminal
  - rosparam list
  - rosparam get /hellocm/cycletime
  - rosparam set /hellocm/cycletime 50
- CarMaker (e.g. for test automation)
  - Currenlty no direct support
  - Tcl’s “exec” command can be used with ScriptControl

3 Examples

3.1 HelloCM

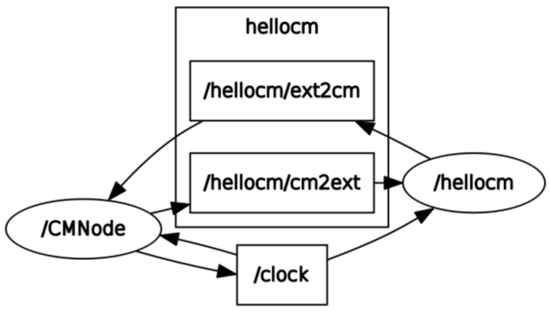


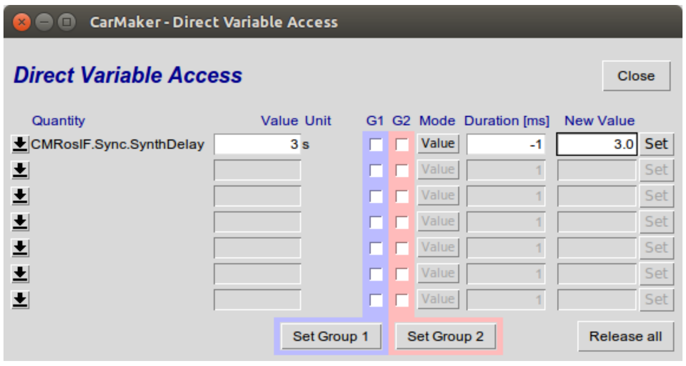
Figure 4: ROS rqt Node Graph for the HelloCM Example

- Simple example for demonstration of communication
- The current example can be used without or with hard synchronization between the CarMaker ROS Node an the external ROS Node
  - 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 Clock Server and provides simulation time to other ROS Nodes
- Related ROS packages with source code, launch files, etc. are located in “<CMProjDir>/ros/ros1_ws/src”

3.1.1 Topic based Synchronization

The general synchronization method is described in chapter “4.8 Process Synchronization (experimental)”.

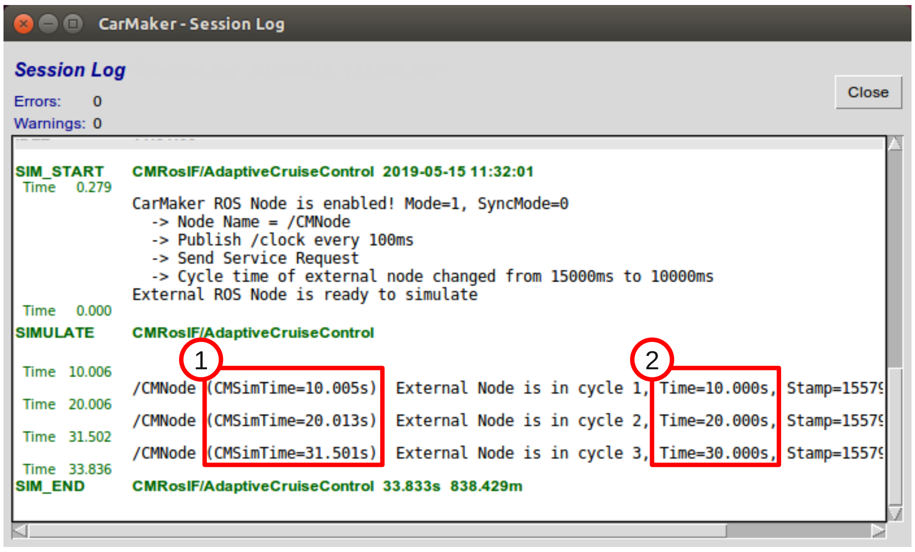
The HelloCM example allows an activation of the synchronization with parameter “Node.Sync.Mode” (see chapter “Parameterization with Infofile”) and provides the User Accessible Quantity “CMRosIF.Sync.SynthDelay” to create an synthetic delay on side of external ROS Node.



Quantity	Value	Unit	G1	G2	Mode	Duration [ms]	New Value	
CMRosIF.Sync.SynthDelay	3	s	☐	☐	Value	-1	3.0	Set
			☐	☐	Value	1		Set
			☐	☐	Value	1		Set
			☐	☐	Value	1		Set
			☐	☐	Value	1		Set
			☐	☐	Value	1		Set
			☐	☐	Value	1		Set

Figure 5: User Accessible Quantity to test synchronization

By default the synchronization is off and results in a behaviour shown in Figure 6. The CarMaker ROS Node acts as 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 CarMaker ROS Node at simulation time (1) with a non-constant delay. The third transmission at 30s was additionally delayed with a synthetic delay of 1s.

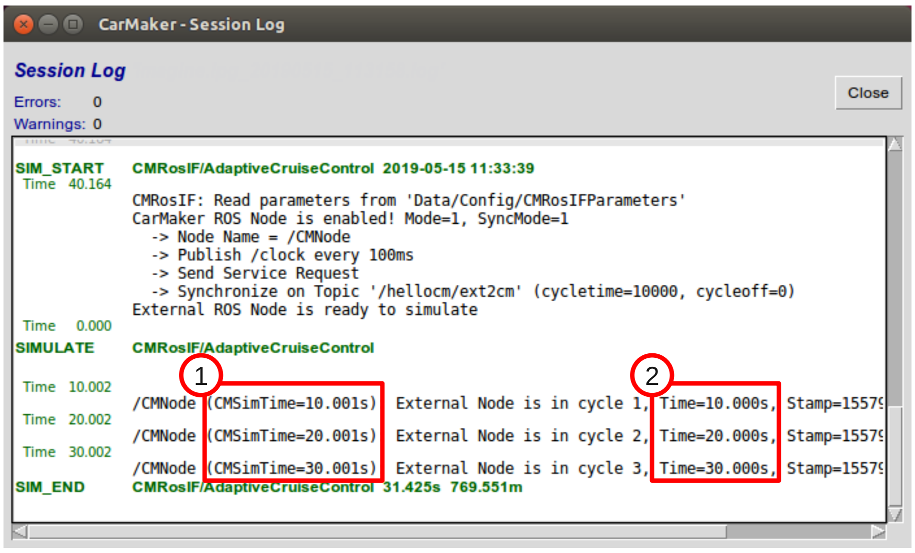


```
SIM_START    CMRosIF/AdaptiveCruiseControl  2019-05-15 11:32:01
 Time   0.279
             CarMaker ROS Node is enabled! Mode=1, SyncMode=0
               -> Node Name = /CMNode
               -> Publish /clock every 100ms
               -> Send Service Request
               -> Cycle time of external node changed from 15000ms to 10000ms
             External ROS Node is ready to simulate
 Time   0.000
SIMULATE     CMRosIF/AdaptiveCruiseControl

 Time  10.006
             /CMNode (CMSimTime=10.005s) External Node is in cycle 1, Time=10.000s, Stamp=1557
 Time  20.006
             /CMNode (CMSimTime=20.013s) External Node is in cycle 2, Time=20.000s, Stamp=1557
 Time  31.502
             /CMNode (CMSimTime=31.501s) External Node is in cycle 3, Time=30.000s, Stamp=1557
 Time  33.836
SIM_END      CMRosIF/AdaptiveCruiseControl  33.833s  838.429m
```

Figure 6: Simulation without synchronization

The same simulation was started with “Node.Sync.Mode = 1” (Figure 7). Here all messages arrive constantly 1ms after the external Node has been triggered.



```
SIM_START    CMRosIF/AdaptiveCruiseControl  2019-05-15 11:33:39
 Time  40.164
             CMRosIF: Read parameters from 'Data/Config/CMRosIFParameters'
             CarMaker ROS Node is enabled! Mode=1, SyncMode=1
               -> Node Name = /CMNode
               -> Publish /clock every 100ms
               -> Send Service Request
               -> Synchronize on Topic '/hellocm/ext2cm' (cycletime=10000, cycleoff=0)
             External ROS Node is ready to simulate
 Time   0.000
SIMULATE     CMRosIF/AdaptiveCruiseControl

 Time  10.002
             /CMNode (CMSimTime=10.001s) External Node is in cycle 1, Time=10.000s, Stamp=1557
 Time  20.002
             /CMNode (CMSimTime=20.001s) External Node is in cycle 2, Time=20.000s, Stamp=1557
 Time  30.002
             /CMNode (CMSimTime=30.001s) External Node is in cycle 3, Time=30.000s, Stamp=1557
SIM_END      CMRosIF/AdaptiveCruiseControl  31.425s  769.551m
```

Figure 7: Simulation with topic based synchronization

4 Documentation

4.1 Installation

- The CMRosIF needs an already existing CarMaker Project directory with included sources/build environment (“src/” folder with CM_Main.c, …) to build the CarMaker executable. If the “src/” folder is missing please use “CM Main GUI -> File -> Project Folder -> Update Project…” with enabled component “Sources/Build Environment”.
- It is recommended to create a backup of your already existing CarMaker Project Directory
- The update procedure is identical for CarMaker, TruckMaker, …
- Copy the files from zip file to the corresponding folders into your already existing CarMaker Project Directory (e.g. by directly extracting inside the directory)

In general following files need to be updated:

- Configure your **"~/.bashrc"** for easier use, e.g. add

```
# Additional paths for CM
addpath ()    { for d in "$@"; do PATH="$PATH:$d"; done; }
addpath /opt/ipg/bin /opt/ipg/hil/linux/bin /opt/ipg/hil/linux/GUI
```

- **ros/ros1_ws/src/hellocm_cmnode/CMakeLists.txt**
    - Update the CarMaker Version string and numerical value
    - The path to CarMaker installation and include folder is updated implicitely
    - e.g. update from old CarMaker version 7.1.2

    ```
    set(CARMAKER_VER 7.1.2)
    set(CARMAKER_NUMVER 70102)
    ```

    - to new CarMaker version 7.1.3

    ```
    set(CARMAKER_VER 7.1.3)
    set(CARMAKER_NUMVER 70103)
    ```

    - For **CarMaker version >= 8.0** addtionally set the variable CARMAKER_DIR to

    ```
    set(CARMAKER_DIR $ENV{IPGHOME}/carmaker/linux64-${CARMAKER_VER})
    ```

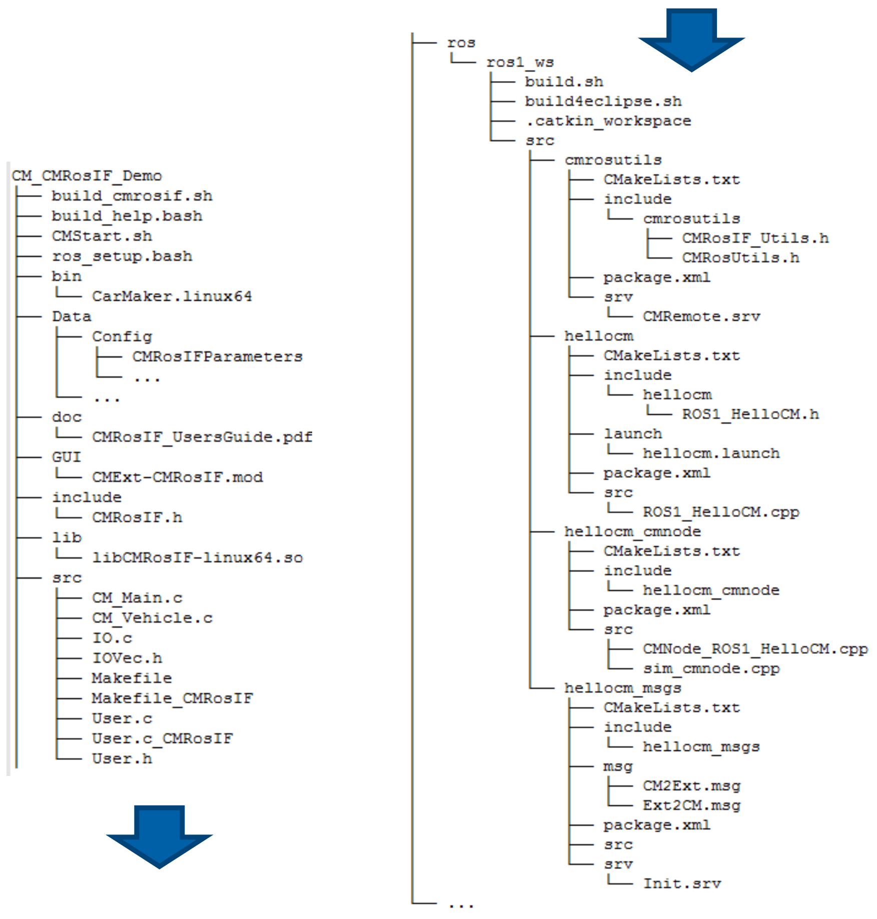
- **src/Makefile**
    - The Makefile rule to build the CarMaker executable needs to be extended that the shared library for the CarMaker ROS Interface can be loaded properly. Please add following lines e.g. **after the LD_LIBS and OBJS_*.block** of the original CarMaker Makefile.

    ```
    # CarMaker extension CMRosIF (provided by IPG)
    DEF_CFLAGS      += -DWITH_CMROSIF
    LD_LIBS_OS      += -lCMRosIF-$(ARCH)
    LDFLAGS         += -L./../lib/
    LDFLAGS         += -Wl,-rpath,'$$ORIGIN/../lib/'
    INC_CFLAGS      += -I../include
    CFLAGS          += -rdynamic
    ```

- **src/User.c**
    - Copy the lines marked with **“WITH_CMROSIF”** from “User.c_CMRosIF” to your User.c”. If not documented otherwise in the CarMaker RealeaseNotes the file can be used directly.
- **CMStart.sh**
    - Update CarMaker GUI version to be started
- Information concerning build procedure and startup are decribed in chapter “2.3 Build necessary Parts” ff.

4.2 Folder/File Overview

The paths below are relative to the CarMaker Project Directory and describe the structure and content after Installation of the files for CMRosIF. The structure can be adapted for different use cases (references in files have to be updated).



```
CM_CMRosIF_Demo
├── build_cmrosif.sh
├── build_help.bash
├── CMStart.sh
├── ros_setup.bash
├── bin
│   └── CarMaker.linux64
├── Data
│   ├── Config
│   │   ├── CMRosIFParameters
│   │   └── ...
│   └── ...
├── doc
│   └── CMRosIF_UsersGuide.pdf
├── GUI
│   └── CMExt-CMRosIF.mod
├── include
│   └── CMRosIF.h
├── lib
│   └── libCMRosIF-linux64.so
├── src
│   ├── CM_Main.c
│   ├── CM_Vehicle.c
│   ├── IO.c
│   ├── IOVec.h
│   ├── Makefile
│   ├── Makefile_CMRosIF
│   ├── User.c
│   ├── User.c_CMRosIF
│   └── User.h
├── ros
│   └── ros1_ws
│       ├── build.sh
│       ├── build4eclipse.sh
│       ├── .catkin_workspace
│       └── src
│           ├── cmrosutils
│           │   ├── CMakeLists.txt
│           │   ├── include
│           │   │   └── cmrosutils
│           │   │       ├── CMRosIF_Utils.h
│           │   │       └── CMRosUtils.h
│           │   ├── package.xml
│           │   └── srv
│           │       └── CMRemote.srv
│           ├── hellocm
│           │   ├── CMakeLists.txt
│           │   ├── include
│           │   │   └── hellocm
│           │   │       └── ROS1_HelloCM.h
│           │   ├── launch
│           │   │   └── hellocm.launch
│           │   ├── package.xml
│           │   └── src
│           │       └── ROS1_HelloCM.cpp
│           ├── hellocm_cmnode
│           │   ├── CMakeLists.txt
│           │   ├── include
│           │   │   └── hellocm_cmnode
│           │   ├── package.xml
│           │   └── src
│           │       ├── CMNode_ROS1_HelloCM.cpp
│           │       └── sim_cmnode.cpp
│           └── hellocm_msgs
│               ├── CMakeLists.txt
│               ├── include
│               │   └── hellocm_msgs
│               ├── msg
│               │   ├── CM2Ext.msg
│               │   └── Ext2CM.msg
│               ├── package.xml
│               ├── src
│               └── srv
│                   └── Init.srv
└── ...
```

Figure 8: Folder/File overview for the example

Description of most relevant folders/files:

- **build_cmrosif.sh**
  - Buids ROS workspace and CarMaker. Experimental for fast ramp up of the examples!
- **CMStart.sh**
  - Start CarMaker GUI and loads GUI extension. “build_cmrosif.sh” has to be executed before (for this example only once)

- **Data/Config/CMRosIFParameters**
  - CarMaker Infofile with parameters for CarMaker ROS Interface and CarMaker ROS Node
    - Enable/Disable Interface/Clock Server
    - Arguments for CarMaker ROS Node for remapping arguments
    - e.g. "Cfg.Args = __ns:=MyNamespace" to push down all Topics, ...
  - Accessible via " CM Main GUI -> Extras -> CMRosIF -> Edit Parameters"
  - More Information see “4.4 Parameterization with Infofile”
- **GUI/**
  - Folder for CarMaker GUI Extension of current CarMaker Project Directory
  - The mod files can be copied to CarMaker installation folder to be available globally
    - CMExt-CMRosIF.mod: see chapter “4.3 CarMaker GUI Extension”
- **include/**
  - Folder with additional include files
  - The header CMRosIF.h for CarMaker ROS Interface is located in this folder
- **src/**
  - CarMaker source folder with files for user accessible modules of the CMRosIF extension
  - **Makefile**
    - Default CarMaker Makefile with additional flags/libraries for CMRosIF
    - Building CarMaker executable with CMRosIF
  - **User.c**
    - Calling functions of CarMaker ROS Interface provided by “lib/.../CMRosIF.so”
    - see "include/CMRosIF.h" for more details
- **ros/ros1_ws/**
  - Native ROS1 catkin workspace with user packages (messages, nodes, …)
  - Including Topics and Services as well as the external ROS Node and shared library for CarMaker ROS Node for the HelloCM example
  - The workspace might be also set as an symlink to an already existing ROS workspace
  - **build.sh** or **build4eclipse.sh**
    - Build scripts for the ROS workspace. While “build.sh” runs just a simple build, build4eclipse.sh prepares the ROS “/build” folder for a eclipse project that can be imported to eclipse (inside eclipse: “File -> Import -> Existing Projects into Workspace”)
  - **src/hellocm**
    - Source code for external ROS node (independent from CarMaker)
  - **src/hellocm_cmnode**
    - Source code for shared library with CarMaker ROS Node
    - Including wrapper functions for CMRosIF
  - **src/hellocm_msgs**
    - Messages and Services used by the nodes above
  - **src/cmrosutils**

    General utilities when using ROS in combination with CarMaker (under development)

4.3 CarMaker GUI Extension

The CarMaker ROS Interface comes up 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 CarMaker Project Directory and loaded via an additional command line argument “-ext <Name of *.mod file”” when starting the CarMaker GUI, e.g. if mod file is located in a folder “<CMProjDir>/GUI” the command is

“**CM . –ext GUI/CMRosIF.mod**” or **CM-7.1.2 . –ext GUI/CMRosIF.mod**

For the CMRosIF example with CarMaker Project Directory, the command line is located in the script “**CMStart.sh**”. It is necessary that the CarMaker installation path has been added to the systems search path (please check chapter **“4.1 Installation”** for additional information).

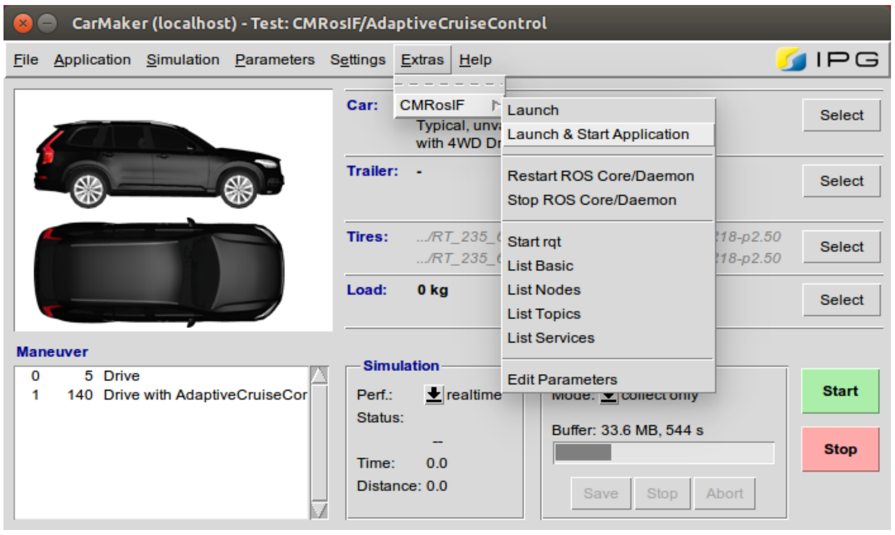


Figure 9: CarMaker GUI Extension

Following commands are currently available:

Launch

This command starts a new terminal, sources the script “<CMProjDir>/ros_setup.bash” and executes the ROS program “roslaunch”. An already opened Terminal (e.g. executing the ROS Core and ROS Nodes) started with a previous “Launch” execution will be stopped if the feature “TerminalCmd” is enabled. The arguments for “roslaunch” can be parameterized in the Infofile “<CMProjDir>/Data/Config/CMRosIFParameters”. (see chapter “4.4 Parameterization with Infofile”)

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 “Parameterization with Infofile” 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 rocscore 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 “4.4 Parameterization with Infofile”)

List <Basic/Nodes/Topics/Services>

The commands run the corresponding ROS programs rostopic, rosnode, 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 “4.4 Parameterization with Infofile”).

If nothing happens it is possible that the default editor is not set correctly. For Ubuntu and before CM6.0.4 there is an additional mod file “CMExt-GUI_PatchUbuntu.mod” that fixes this issue starting “gedit”. If another Editor should be used the Tcl variable “$Pgm(TextEditor.linux)” can be manipulated e.g. via ScriptControl (current session only, e.g. “set ::Pgm(TextEditor.linux) gedit”), an entry with the same command in the file “.CarMaker.tcl” located in the root directory of the CarMaker Project Directory or your home directory.

4.4 Parameterization with Infofile

Following paramaters can be used to parameterize the ROS Independent CarMaker ROS Interface. The parameters allow a modification of internal mechanism and e.g. arguments provided to CarMaker ROS Node for the function “CMRosIF_CMNode_Init()”.

The parameters are managed in different groups with 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. Depending on Node design!

4.4.1 General Configuration

Cfg.Lib.Path

This parameter is not optional!

Path to the shared library with CarMaker ROS Node. The path can be absolute or relative to current CMProjDir. The shared library is currently loaded at startup of the CarMaker executable.

Example:

“Cfg.Lib.Path = ./ros/ros1_ws/install/lib/libCMNode_ROS1_HelloCM.so“

Cfg.Mode

This parameter is optional! Default = 1

The CarMaker ROS Interface can be disabled (Cfg.Mode=0). In this case the shared library with CarMaker ROS Node is not loaded after starting the CarMaker executable and no CarMaker ROS Node will be available. Standard functions provided by “CMRosIF.h” can still be called but will not have any effect. User functions received with “CMRosIF_GetSymbol()” will be NULL pointers and have to be managed by the user (e.g. “if (MyFunc != NULL) MyFunc();”)!

Cfg.Name

This parameter is optional! Default = “CMNode”

Name of the CarMaker ROS Node provided as argument by “CMRosIF_CMNode_Init()”. The string is normally provided to “ros::init()” inside CarMaker Ros Node. The naming will be affected by the remapping arguments that can be provided with the “Cfg.Args” infofile parameter (e.g. to push the whole node into own namespace).

Cfg.Args

This parameter is optional! Default = “”

The parameter string is mapped to the Argc and Argv arguments provided by the function “CMRosIF_CMNode_Init()” and can be provided to the ROS function “ros::init()”. This allows the user providing arguments in a style of a standalone ROS node executable 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:

If “Cfg.Args = __ns:=MyVhcl” “CMRosIF_CMNode_Init()” will provide Argc=2 and e.g. Argv[1]= “__ns:=MyVhcl”. When provided to “ros::init()” Node name, Topics, etc. will be pushed down to the ROS namespace “/MyVhcl” (e.g. Node name “/CMNode” will be “/MyVhcl/CMNode”.

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)
  - 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)
  - Mechanism to handle multiple Terminals
  - Automatic termination of launch terminal
  - Optional multi terminal/tab mechanism
    - Managing different tabs and terminals and run multiple launch files in same (one tab for each) or multiple terminals
    - Create additional terminals with Linux commands
    - Check "Cfg.UserUtils.FPath" and "Cfg.LaunchProc"
  - **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

4.4.2 Launchfile and rqt

Launch.Args

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

4.4.3 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.
e.g. 0=Off, 1=Enabled, 2=ThreadedSpin, … .

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 an node internal synchronization mode.
e.g. 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 stops 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 ROS Clock Server by setting the ROS parameter “/use_sim_time” and publishing the current simulation time every “Node.nCyclesClock” cycles.

Node.nCyclesClock

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

Number of CarMaker cycles. Generally duration for 1 cycle is 1ms.

4.5 Build Process

Chapter “2.3 Build necessary Parts” shows how to build the example with a special script file. All the steps can be executed by hand as described below.

4.5.1 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 install” (default behaviour). All packages located in “src” folder below the workspace will be built. A package can be ignored by adding the file “CATKIN_IGNORE” 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 be also 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 http://wiki.ros.org/catkin/workspaces.

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

4.5.2 Shared Library with CarMaker ROS Node

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

- The path to the shared library with CarMaker ROS Node dynamically loaded inside CarMaker executable can be adapted by editing the parameter “Cfg.Lib.Path” (see 4.4.1 General Configuration)
- The source code of the shared library contains C functions that are not allowed to be renamed (otherwise the Interface won’t find correct symbols)
- Additional user defined functions can be created inside the library and used e.g. inside User.c via function pointers (see example code for CMNode, User.C and “CMRosIF.h”)
- Code to CarMaker Data (e.g. Sensors and Vehicle Model) can be directly accessed inside the code for the shared library with CarMaker ROS Node. Just add the necessary header from “<CMInstDir>/include”. Please note that the shared library will have dependency to the CarMaker libraries and CarMaker Version at compile time! For a new CarMaker Version (especially new major release) 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 CMNode package for more information.

4.5.3 CarMaker Executable with CarMaker ROS Interface

The CarMaker executable with CarMaker ROS Interface is built with the CarMaker Makefile, e.g. located in “<CMProjDir>/src” and can be built by opening a new terminal and executing “make” inside this folder (more information see “Programmers Guide”). The CarMaker executable for this example normally only needs to be compiled after modifications in CarMaker user modules (User.c, …) or when the CMRosIF API has changed.

The CarMaker ROS Interface currently consists of a header e.g. “<CMProjDir>/include/CMRosIF.h” and a shared library e.g. “<CMProjDir>/lib/<platform>/<CMVersion>/libCMRosIF.so”. The shared library is linked with the CarMaker executable and is automatically loaded at startup.

4.6 Interaction of CarMaker and Shared Library with CarMaker ROS Node

The shared library with CarMaker ROS Node is not directly linked to CarMaker executable. When calling the initialization function for the CarMaker ROS Interface e.g. inside “User.c” the shared library with CarMaker ROS Node parameterized by “Cfg.Lib.Path” (see 4.4 Parameterization with Infofile) will be loaded during runtime of the CarMaker executable. Commonly used default functions are already existing (see “CMRosIF.h”). These functions call internal function pointers. These function pointers get the related functions from the shared library with CarMaker ROS Node.

For example, the function “***CMRosIF_TestRun_Start_atBegin()***” from the ***CMRosIF*** will call “***CMRosIF_CMNode_TestRun_Start_atBegin()***” from the shared library with CarMaker ROS Node.

The “***CMRosIF_Init()***” has to be called before any other function from the CMRosIF API.

Additional user functions can be defined inside the shared library with CarMaker ROS Node within the “***extern “C”***” block. The function “***CMRosIF_GetSymbol()***” allows to search for functions/Symbols inside the dynamically loaded library with CarMaker ROS Node and returns a function pointer that can be used e.g. in User.c. Check “User.c” and source code for shared library 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, …) located in the CarMaker installation directory (e.g. “/opt/ipg/hil/linux/include”) to the shared library with CarMaker ROS Node. Libraries for linking are located in the “lib/” folder of the CarMaker installation directory.

4.7 Job Mechanism (experimental)

The job mechanism can be used to execute one or more cyclic task (e.g. publish topic, copy data from message buffer to vehicle model, …) dependent on e.g. the CarMaker cycle number relative to simulation start. This allows us to pusblish data e.g. every 10ms with an offset of e.g. 5ms.

Currently two different modes are available

- "Default Mode"
- "Extended Mode"

While the "Default Mode" can be used to realize cyclic tasks as mentioned above, the extended mode expects a preparatory task to be finished before the actual task can start.

The basic workflow for the job mechanism is

1. Create a job handle via CMCRJob_Create()
2. Initialize job via CMCRJob_Init()
3. Optional (Only for "Extended Mode")
   a. Check every simulation cycle if a preparation for this job needs to be done in current cycle via CMCRJob_DoPrep()

      Optional: Manual confirmation that all preparation tasks related to this job have been finished via CMCRJob_DoPrep_SetDone()

   b.
4. Check every simulation cycle if a job needs to be done in current cycle via CMCRJob_DoJob()
   a. Optional: Manual confirmation that all tasks related to this job have been finished via CMCRJob_DoJob_SetDone()
5. Reinitialize for next simulaton loop via CMCRJob_Init() or delete via CMCRJob_Delete()

The job mechanism is independent from ROS and **only available inside the CarMaker ROS Node**. Please check the header "/ros/ros1_ws/src/cmrosutils/include/cmrosutils/CMRosIF_Utils.h" for more information concering available functions and the ROS package "hellocm_cmnode" for general usage!

4.8 Process Synchronization (experimental)

By default ROS is a non-deterministic software framework that transfers messages between several Nodes using the publish/subscribe mechanism. The effective time until a message is received depends on the system workload and transmission path. So multiple processes run independent that may results in non-reproducible simulations. If running the simulaiton in soft real time the effect might be small and is acceptable for many use cases. But running CarMaker with maximum simulation speed the relative time until a message is transferred is relevant for the simulation result.

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 a free parameterizable message is received. The synchronization is based on an internal cycle counter and the knowledge of expected cycle time of the external Node inside the CarMaker Node.

The mechanism can be used in different ways

1. Simulation time based synchronization
    - The ROS “/use_sim_time” mechanism needs to be activated
    - CarMaker ROS Node has to act as clock server
    - ROS Timer inside extrernal ROS Node that reacts on changes in simulation time
    - Provide cycle time and topic name for synchronization to CarMaker ROS Node
2. Data triggered synchronization
    - ROS Node interaction with one or more calculation chains, each chain finishes with topic in CarMaker ROS Node that is used for synchronization
    - One ore more subscription(s) in external ROS Node(s)
        - Includes algorithm, etc.
        - ROS publish at end of calculation chain
    - Provide cycyle time and topic name for synchronization to CarMaker ROS Node
    - 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 “HelloCM”. The effect of synchronization is shown in chapter “Topic based Synchronization”.

5 Release History

5.1 Version 0.6.8

General

- Updatet User.c and Makefile for CarMaker executable in “<CMProjDir>/src/”
- Default target for ROS build and examples is “devel” (before “install”)
- Added script “<CMProjDir>/ros/ros1_ws/src/build4eclipse.sh”
  - Builds a catkin workspace with eclipse project files inside the “build” folder and allows a direct import as eclipse project
  - According to “Catkin-y approach” described on http://wiki.ros.org/IDEs#Eclipse
  - 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 für 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”
  - Manages terminal started via CarMaker GUI e.g. menu entry “CM Main GUI -> Extras -> CMRosIF-> Launch”
  - More information see chapter “Parameterization with Infofile” and parameter “Cfg.Features”