Car2X Messages

The „CARP protocol “was extended by a set of messages to let the NIOS2 processor system fit into the role of the communication central of the car. One NIOS2 core is handling the internal state of the car and the other one is in charge of all the communication between internal car parts and external communication partners like other cars or “car2x” stations. All the internal communication is based on the work of Florian Hisch and has nearly not been modified. The only change was a role switch in terms of server-client relationship. Unlike before, the central unit now is a server, external clients can connect to.

Keeping that in mind, there is an external communication interface featuring the following messages as an extension of the “CARP protocol”. Each message will be answered by the car after being processed or outdated. By parsing in the messages in the “socketserver.cpp” file directly from the incoming TCP/IP byte stream, there is a new message object created for every new message.

Directly after that the “sss\_exec\_command()” function handles the received message by reading out the message type an then taking various steps depending on the specific message.

In case of simple “polling messages” that don’t interact with the car state, like the “CInfoStateMessage” and the “CInfoSensorMessage” which do only read some information out of the current state, there is immediately created and sent an answer message, containing the required data.

The other three messages have an influence on the car state and therefore have to be queued until the car state gets updated. Once the update is there, main loop of the socketserver checks if the requested state change like for example an emergency brake process has been performed or not, deletes the message from the queue and sends an answer message. Depending on this check the produced answer message contains a different flag:

* “A” for successful execution
* “F” for failed execution
* “O” for the message being outdated

The main challenge and reason for this check is the fact that communication and state control are performed by different NIOS2 cores. While the state is being updated periodically, incoming messages might arrive more frequent. If for example 3 “CControlMessages”, which contain the information to set the motors to a specific speed, are received within one state update cycle, it is obvious that only the last one should be executed in the new state and the older ones get outdated as soon as a new message of the same type is received. Whenever a message gets queued, and there is already a message in the queue, the outdated one gets immediately answered with an “O” flag. It is outdated and doesn’t have to be executed any more, since it has been overwritten by the latest message, resulting in only one queued message at maximum for every message type at every state update. For a more detailed description of the queueing process see next chapter.

* **CEmergencyBrakeMessage**

Example packet sent to the car:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | PacketNumber(16bit) | PayloadLength(16bit): 0x04 | Type: 0x20 | Length: 0x04 | Subtype: 0x0 | Flags:  0x0 |

After receiving this message, the control core is requested to change the car state into emergency braking and this message is queued to wait for the next state update, or being outdated.

Example answer message sent by the car:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | ControlCoreCounter (32bit) | CommCoreCounter(32bit) | PayloadLength (32bit) | Success flag (‘A’,’F’,’O’) | Type: 0x20 | PacketNumber of the received message(16bit) |

* **CControlMessage**

Example packet sent to the car:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | PacketNumber(16bit) | PayloadLength(16bit): 0x08 | Type: 0x30 | Length: 0x08 | Subtype: 0x0 | Flags:  0x0 | V1 | V2 | V3 | V4 |

After receiving this message, the control core is requested to set the specified motor velocities, specified in the payload. It has to be mentioned that in case the sender of this message is not registered as the current source of control, the message is not queued but immediately replied as failed because the sender is not allowed to control the car.

Afterwards this message is queued to wait for the next state update, or being outdated.

Example answer message sent by the car:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | ControlCoreCounter (32bit) | CommCoreCounter(32bit) | PayloadLength (32bit) | Success flag (‘A’,’F’,’O’) | Type: 0x30 | PacketNumber of the received message(16bit) | V1 | V2 | V3 | V4 |

V1…4 are now standing for the current desired motor speed values delivered to the specific PWM motor controllers. In case of a successful message, they are equal to the requested values of the message, sent to the car, otherwise they differ and the message is answered as “failed”, giving the controlling unit feedback about the current state of the car, to let it know what might have failed.

* **CRemoteControlMessage**

Example packet sent to the car:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | PacketNumber(16bit) | PayloadLength(16bit): 0x08 | Type: 0x60 | Length: 0x08 | Subtype: 0x0 | Flags:  0x0 | IP1 | IP2 | IP3 | IP4 |

The payload of this message contains the IP of a source which should be allowed to control the car by using “CControlMessages”. Per default this source is the unit inside of the car featuring the ImageProcessing with its IP 10.10.100.110. In the standard state of the car “AutoDrive”, the car is controlled from this IP by “CControlMessages”. By sending a “CRemoteControlMessage” to the car containing a different IP, the car is requested to set its state to “ManualDrive” with the new source IP locked for “CControlMessages”. Sending 0.0.0.0 as new IP will set the car back to “AutoDrive”, using the ImageProcessing IP again.

As before, the answer gets queued until the state is updated for the next time.

Example answer message sent by the car:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | ControlCoreCounter (32bit) | CommCoreCounter(32bit) | PayloadLength (32bit) | Success flag (‘A’,’F’,’O’) | Type: 0x60 | PacketNumber of the received message(16bit) | IP1 | IP2 | IP3 | IP4 |

IP1…4 are the current state values of the locked IP. In case of success they equal the requested one, otherwise they give the feedback about who is currently controlling the car.

* **CInfoStateMessage**

Example packet sent to the car:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | PacketNumber(16bit) | PayloadLength(16bit): 0x04 | Type: 0x40 | Length: 0x04 | Subtype: 0x0 | Flags:  0x0 |

This message just polls the current state information from the car. It is answered immediately and thus has not to be queued. The answer always contains the “A” flag for success and features the biggest part of the state object from the shared memory as its payload. It can be used for debugging purposes or by the ImageProcessing unit or a station as additional odometry feedback.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | ControlCoreCounter (32bit) | CommCoreCounter(32bit) | PayloadLength (32bit) | Success flag (‘A’,’F’,’O’) | Type: 0x40 | PacketNumber of the received message(16bit) | StateStruct(sizeof(stateStruct)) |

As “ControlCoreCounter” and ”CommCoreCounter” are already part of the answer, they are excluded from the payload and since the sensor values are polled by the “CInfoSensorMessage”, they are also excluded to safe bandwidth.

* **CInfoSensorMessage**

Example packet sent to the car:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | PacketNumber(16bit) | PayloadLength(16bit): 0x04 | Type: 0x50 | Length: 0x04 | Subtype: 0x0 | Flags:  0x0 |

This message just polls the current sensor information from the car. It is answered immediately and thus has not to be queued. The answer always contains the “A” flag for success and features the whole sensor information which are part of the car state. This message can be used to access the sensor data.

Example answer message sent by the car:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | A | R | P | ControlCoreCounter (32bit) | CommCoreCounter (32bit) | PayloadLength (32bit): 0x08 | Success flag ‘A’ | Type: 0x50 | PacketNumber of the received message(16bit) | Sensor 1 Value(32bit) | Sensor 2 Value(32bit) |