Week 9: Creating the Autoware <-> CAN DBW (Dataspeed) Vehicle Interface

Resources:

- Vehicle Interface Overview
- Creating the Vehicle Interface
- Ackermann Kinematic Model
- Customizing for Differential Drive Model

As of 11/22/2025, the relevant ROS2 messages (msgs) between Autoware and the Dataspeed DBW are as follows:

For DBW CAN (Dataspeed) to Autoware

- **dbw_ford_msgs.msg** - importing TwistStamped, Misc1Report, SteeringReport as SteeringReport_dbw, GearReport as GearReport_dbw
- **autoware_vehicle_msgs.msg** - importing VelocityReport, TurnIndicatorsReport, GearReport, SteeringReport, ControlModeReport, HazardLightsReport

For Autoware to DBW CAN

- **dbw_ford_msgs.msg** - importing GearCmd, BrakeCmd, SteeringCmd, ThrottleCmd, MiscCmd
- **dataspeed ulc_msgs.msg** - importing UlcCmd (this is for the velocity reported by the Dataspeed DBW)
- **autoware_vehicle_msgs.msg** - importing TurnIndicatorsCommand, GearCommand, HazardLightCommand, and VelocityReport
- **autoware_control_msgs.msg** - importing Control (this single file will import ControlHorizon.msg, Lateral.msg, Longitudinal.msg) this message is not in the pipeline figure in the previous page as the documentation provided by Autoware is deprecated. This replaces autoware_auto_vehicle_msgs.msg
- **tier4_vehicle_msgs.msg** - importing VehicleEmergencyStamped

With these relevant messages, an MKZ Autoware <-> Dataspeed interface package can be made. It will also be provided but it is best to check that the messages within the packages align with the ROS2 messages in the Autoware and Dataspeed DBW package.

The provided MKZ Dataspeed <-> Autoware interface package will be enough for mapped driving but it can be advanced further with Ackermann kinematic modeling and differential drive vehicles. Please refer to the Autoware documentation for those instructions.

MKZ_Interface (ROS2 Package) - drop this package into your Autoware directory, specifically: “autoware/src/sensor_component/external” then build using the following command

```
cd autoware   # entering our Autoware directory
colcon build --packages-up-to mkz_interface
```

This will (re)build the other packages that the interface is dependent on.

To launch the interface, run the following command:

```
ros2 launch mkz_interface mkz_interface.launch.xml
```

The “mkz_interface.launch.xml” is a simple launch file that runs both the Autoware-to-DBW and DBW-to-Autoware nodes in one terminal session. If the messages and correct topics were set up in the node files, the terminal should look like below:

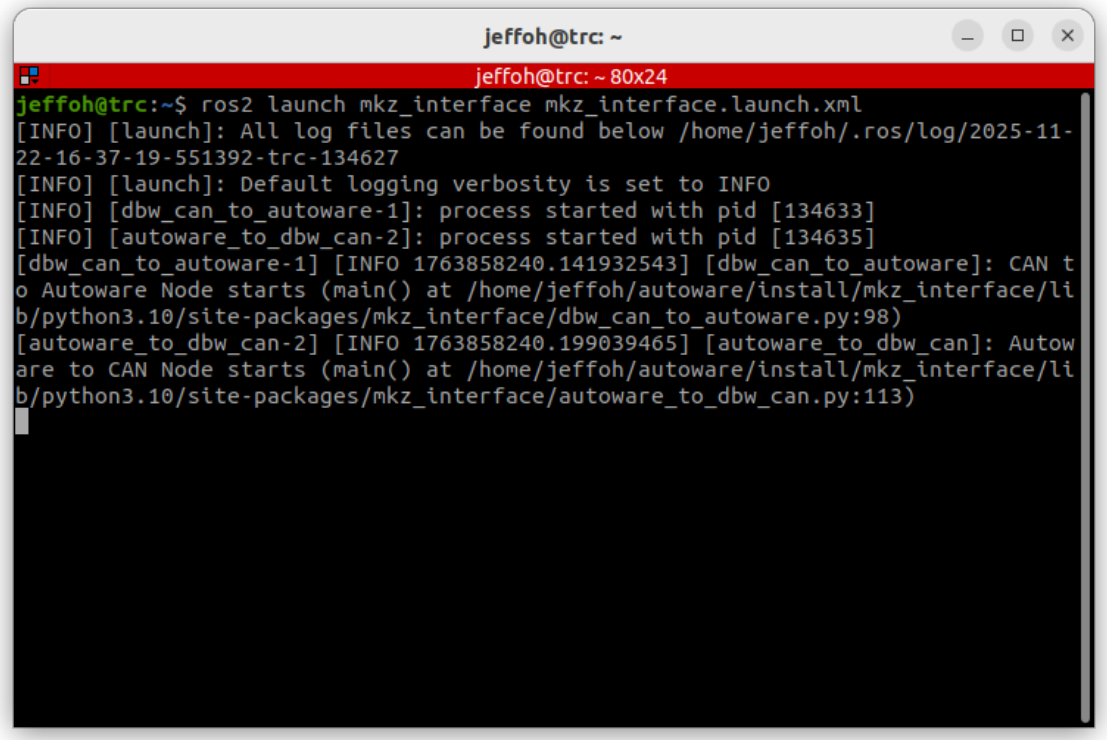

Figure 1: mkz_interface running correctly

Below are the two nodes within the package that convert DBW Dataspeed -> Autoware and Autoware -> DBW Dataspeed for reference:

autoware_to_dbw_can.py

```
import rclpy
from rclpy.node import Node
import math
from std_msgs.msg import Bool, Empty
from geometry_msgs.msg import Twist
from dbw_ford_msgs.msg import GearCmd, BrakeCmd, SteeringCmd, ThrottleCmd, MiscCmd
from dataspeed_ulc_msgs.msg import UlcCmd
import dataspeed_ulc_msgs, dbw_ford_msgs
from autoware_vehicle_msgs.msg import TurnIndicatorsCommand, GearCommand,
HazardLightsCommand, VelocityReport
from autoware_control_msgs.msg import Control
from tier4_vehicle_msgs.msg import VehicleEmergencyStamped


class Autoware_to_Dbw_can(Node):
    def __init__(self):
        super().__init__('autoware_to_dbw_can')

        # Publishers
        self.pub_ulc = self.create_publisher(UlcCmd,'/vehicle/ulc_cmd', 10)  #for longitudinal speed
        #self.pub_brake = self.create_publisher(BrakeCmd,'/vehicle/brake_cmd', 10)
        #self.pub_throttle = self.create_publisher(ThrottleCmd,'/vehicle/throttle_cmd', 10)
        self.pub_misc = self.create_publisher(MiscCmd,'/vehicle/misc_cmd', 10) #for turn signal
        #self.pub_gear = self.create_publisher(GearCmd,'/vehicle/status/gear_cmd', 10)
        self.pub_steering = self.create_publisher(SteeringCmd,'/vehicle/steering_cmd', 10)
        #self.pub_control_mode = self.create_publisher(Empty,'/vehicle/enable', 10)

        # Subscribers
        self.subscription_control = self.create_subscription(Control, '/control/command/control_cmd',
self.callback_control, 10)
        self.subscription_velocity = self.create_subscription(VelocityReport, '/vehicle/status/velocity_status',
self.callback_velocity, 10)
        #self.subscription_emergency = self.create_subscription(VehicleEmergencyStamped,
'/control/command/emergency_cmd', self.callback_emergency, 10)
        #self.subscription_gear = self.create_subscription(GearCommand, '/control/command/gear_cmd',
self.callback_gear, 10)
        #self.subscription_hazard= self.create_subscription(HazardLightsCommand,
'/control/command/hazard_lights_cmd', self.callback_hazard, 10)
        self.subscription_turn_signal = self.create_subscription(TurnIndicatorsCommand,
'/control/command/turn_indicators_cmd', self.callback_turn_signal, 10)


        self.misc_msg = MiscCmd()
        self.gear_msg = GearCmd()
        self.wheel_base = 2.8498
        self.steering_ratio = 14.8
        self.current_velocity = 0
```

```
    def callback_control(self, data):
        ulc_msg = UlcCmd()
        ulc_msg.header.stamp = data.stamp
        ulc_msg.header.frame_id = "base_link"
        ulc_msg.clear = True
        ulc_msg.enable_pedals = True
        ulc_msg.enable_steering = False
        ulc_msg.enable_shifting = True
        ulc_msg.shift_from_park = True

        ulc_msg.pedals_mode = dataspeed_ulc_msgs.msg.UlcCmd.SPEED_MODE
        ulc_msg.accel_cmd = 0.0
        ulc_msg.coast_decel = False
        ulc_msg.steering_mode = dataspeed_ulc_msgs.msg.UlcCmd.YAW_RATE_MODE

        ulc_msg.linear_velocity = data.longitudinal.speed
        #ulc_msg.yaw_command = (math.tan(data.lateral.steering_tire_angle) * self.current_velocity ) /
self.wheel_base

        ulc_msg.linear_accel = 0.0
        ulc_msg.linear_decel = 0.0
        ulc_msg.angular_accel = 0.0
        ulc_msg.lateral_accel = 0.0
        ulc_msg.jerk_limit_throttle = 0.0
        ulc_msg.jerk_limit_brake = 0.0

        self.pub_ulc.publish(ulc_msg)

        steering_msg = SteeringCmd()
        steering_msg.header.stamp = self.get_clock().now().to_msg()
        steering_msg.enable = True
        steering_msg.clear = True
        steering_msg.ignore = False
        steering_msg.count = False
        steering_msg.cmd_type = dbw_ford_msgs.msg.SteeringCmd.CMD_ANGLE
        steering_msg.steering_wheel_angle_cmd = data.lateral.steering_tire_angle * self.steering_ratio
        steering_msg.steering_wheel_angle_velocity = 0.0
        self.pub_steering.publish(steering_msg)

    def callback_velocity(self, data):
        self.current_velocity = data.longitudinal_velocity

    def callback_emergency(self, data):
        pass

    def callback_gear(self, data):
        self.gear_msg.header.stamp = self.get_clock().now().to_msg()
        conversion_dict = {0:0, 1:3, 2:4, 20:2, 22:1, 23:5}
        self.gear_msg.cmd.gear = conversion_dict[data.command]

    def callback_hazard(self, data):
```

```
        pass


    def callback_turn_signal(self, data):
        self.misc_msg.header.stamp = self.get_clock().now().to_msg()
        conversion_dict = {0:0,1:0, 2:1, 3:2}
        self.misc_msg.cmd.value = conversion_dict[data.command]
        self.pub_misc.publish(self.misc_msg)

    def publish_all_msgs(self):
        self.pub_brake.publish(self.brake_msg)
        self.pub_throttle.publish(self.throttle_msg)
        self.pub_misc.publish(self.misc_msg)
        self.pub_gear.publish(self.gear_msg)
        self.pub_steering.publish(self.steering_msg)
        self.pub_control_mode.publish(self.control_mode_msg)


def main(args=None):
    rclpy.init(args=args)
    converter_node = Autoware_to_Dbw_can()
    converter_node.get_logger().info("Autoware to CAN Node starts")

    rclpy.spin(converter_node)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    converter_node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

dbw_can_to_autoware.py

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Bool
from geometry_msgs.msg import TwistStamped
from dbw_ford_msgs.msg import Misc1Report
from dbw_ford_msgs.msg import SteeringReport as SteeringReport_dbw
from dbw_ford_msgs.msg import GearReport as GearReport_dbw
from autoware_vehicle_msgs.msg import VelocityReport, TurnIndicatorsReport, GearReport,
SteeringReport, ControlModeReport, HazardLightsReport

class Dbw_can_to_Autoware(Node):
    def __init__(self):
```

```
        super().__init__('dbw_can_to_autoware')

        # Publishers
        self.pub_velocity = self.create_publisher(VelocityReport,'/vehicle/status/velocity_status', 10)
        self.pub_turn_signal =
self.create_publisher(TurnIndicatorsReport,'/vehicle/status/turn_indicators_status', 10)
        self.pub_hazard_signal = self.create_publisher(HazardLightsReport,
'/vehicle/status/hazard_lights_status', 10)
        self.pub_gear = self.create_publisher(GearReport,'/vehicle/status/gear_status', 10)
        self.pub_steering = self.create_publisher(SteeringReport,'/vehicle/status/steering_status', 10)
        self.pub_control_mode = self.create_publisher(ControlModeReport,'/vehicle/status/control_mode', 10)

        # Subscribers
        self.subscription_twist = self.create_subscription(TwistStamped, '/vehicle/twist', self.callback_twist, 10)
        self.subscription_turn_signal = self.create_subscription(Misc1Report, '/vehicle/misc_1_report',
self.callback_turn_signal, 10)
        self.subscription_gear = self.create_subscription(GearReport_dbw, '/vehicle/gear_report',
self.callback_gear, 10)
        self.subscription_steering = self.create_subscription(SteeringReport_dbw, '/vehicle/steering_report',
self.callback_steering, 10)
        #self.subscription_control_mode = self.create_subscription(Bool, '/vehicle/dbw_enabled',
self.callback_control_mode, 10)

        #self.timer = self.create_timer(0.1, self.publish_all_msgs)
        self.steering_ratio = 14.8

        self.reverse = False

    def callback_twist(self, data):
        msg = VelocityReport()
        msg.header = data.header
        msg.longitudinal_velocity = data.twist.linear.x
        if self.reverse:
            msg.longitudinal_velocity = - data.twist.linear.x
        msg.lateral_velocity = data.twist.linear.y
        msg.heading_rate = data.twist.angular.z

        self.pub_velocity.publish(msg)

        ctrl_msg = ControlModeReport()
        ctrl_msg.stamp = self.get_clock().now().to_msg()
        ctrl_msg.mode = 1

        self.pub_control_mode.publish(ctrl_msg)

    def callback_turn_signal(self, data):
        turn_msg = TurnIndicatorsReport()
        turn_msg.stamp = self.get_clock().now().to_msg()
        # msg.stamp = data.header.stamp
        conversion_dict = {0:1, 1:2, 2:3}
        turn_msg.report = conversion_dict[data.turn_signal.value]
        self.pub_turn_signal.publish(turn_msg)
```

```
        hazard_msg = HazardLightsReport()
        hazard_msg.stamp = turn_msg.stamp
        if data.turn_signal.value == 3:
            hazard_msg.report = 2
        else:
            hazard_msg.report = 1

        self.pub_hazard_signal.publish(hazard_msg)

    def callback_gear(self, data):
        msg = GearReport()
        if data.state.gear == 2:
            self.reverse = True
        else:
            self.reverse = False
        msg.stamp = self.get_clock().now().to_msg()
        conversion_dict = {0:0,1:22,2:20,3:1,4:2, 5:23}
        msg.report = conversion_dict[data.state.gear]

        self.pub_gear.publish(msg)

    def callback_steering(self, data):
        msg = SteeringReport()
        msg.stamp = data.header.stamp
        msg.steering_tire_angle = data.steering_wheel_angle / self.steering_ratio

        self.pub_steering.publish(msg)

    def callback_control_mode(self, data):
        msg = ControlModeReport()
        msg.stamp = self.get_clock().now().to_msg()
        msg.mode = 1

        self.pub_control_mode.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    converter_node = Dbw_can_to_Autoware()
    converter_node.get_logger().info("CAN to Autoware Node starts")

    rclpy.spin(converter_node)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    converter_node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
```

```
main()
```