ETCARS Documentation

Release 0.14

Josh Menzel

Apr 06, 2019

Contents

CHAPTER 1

Introduction

1.1 General

ETCARS is an open source telemetry implementation for American Truck Simulator and Euro Truck Simulator 2 created by SCS Software. ETCARS stands for Electronic Truck Communications Addressing and Reporting System. It is built using c++ and officially compatible with Linux and Windows, with the potential to be built on Mac OS X. ETCARS runs a local TCP Socket Server on the computer running the game. This allows for data to be broadcast to any clients that wish to connect. Starting with version 0.14, ETCARS will only have 64-bit support. This is due to the issues encountered when trying to build for 32-bit platforms.

1.2 Built With

ETCARS is built with several other libraries:

- Boost C++ 1.66
- CryptoPP(Crypto++)
- OpenSSL
- cURL
- Steamworks
- SCS SDK

1.3 Contributors

Contributors can be found on the Gitlab repository.

1.4 License

This software is licensed under the GNU GPL v3.0 License.

CHAPTER 2

Getting Started

2.1 Windows

To install ETCARS on Windows, there are some prerequisites to installing:

- Valid OS
- American Truck Simulator and/or Euro Truck Simulator 2 (Steam Linked)
- Microsoft Visual C++ Redistributable 2017

2.1.1 Supported Windows Versions

The following Windows Versions are supported:

- Windows 7 x64
- Windows 8 x64
- Windows 8.1 x64
- Windows 10 x64

2.1.2 Microsoft Visual C++ Redistributable 2017

The Microsoft Visual C++ Redistributable 2017 is a dependency for the plugin to function properly in the Windwos environment. For 64-bit users, you'll need to install both the 32-bit(x86) and 64-bit(x64) version of the redistributable. They can be downloaded using the links below:

- x86
- x64

2.1.3 Installing

Once the Microsoft Visual C++ Redistributable 2017 has been installed, you can now proceed to installing ETCARS! To install ETCARS, simply download one of the installers from the ETCARS website at https://etcars.menzelstudios.com. Then run the installer and boom! You're all set to go!

2.2 Linux

To install the latest version of ETCARS on linux, open a terminal window in a directory you have write access to. Then download the installation script by pasting the following line into the terminal window, then press enter.

```
wget https://gitlab.com/jammerxd/ETCARS/blob/master/ETCARS/install_etcars_linux.sh
```

The installation script will use wget to download 2 files. One is the shared object plugin, the other is a file used by cURL. To run the installation script, execute the following by copying and pasting the following line into the terminal window and pressing enter.

```
bash install_etcars_linux.sh
```

Congratulations! ETCARS is now installed and will run when your game runs!

2.2.1 Additional Information

The current install script is set to install the 64-bit edition ONLY. The 32-bit edition is available for download. The shared object file needs to be placed in the steamapps/common/GAME/bin/linux_x86/plugins folder for 32-bit and steamapps/common/GAME/bin/linux_x64/plugins for 64-bit. | | NOTE: ETCARS uses a certificate bundle for host verification. The file ca-certificates.crt can be downlocaded from the cURL web site as cacert.pem and then renamed to ca-certificates.crt and placed in /etc/ssl/certs/. This is required for cURL and ETCARS to work correctly.

2.3 Mac OS X

Currently OS X is not supported, although it may be added in the future. According to the SDK Examples included from SCS, it may be possible for plugins to run. However our development team currently does not have a Mac system to build and/or test with. You can make requests on the SCS forums or on the Menzel Studios forums.

CHAPTER 3

Receiving Data

To receive data from the game, simply connect to the TCP Socket. The plugin uses port 30001 for clients to connect on.

There are several events and different messages that are broadcast with different structures.

Apart from events, data is broadcast regularly at 1 second intervals.

3.1 Putty

Putty is a basic shell program that can connect to various levels of protocols and is available free of charge at http://putty.org.

To view the raw data, you can open a putty window on the same machine with the game running. The host is `localhost` and the port is `30001`. The connection type is raw.

Then you can see the data as it's broadcast in real time. You may notice the data is in JSON format which is easily decodable in multiple languages.

To make it look pretty, you can copy all of it to the clipboard, then use a website like JSON Lint to make it look nice.

3.2 Programming

To receive data from ETCARS, your programming language should connect using standard TCP socket.

The host is `localhost` and the port is `30001`.

Once connected, you should look to your socket receive or async receive method. This will capture the data broadcast.

3.3 Third Party Packages

There are many other languages out there that have the ability to connect and receive ETCARS's data. Note that none of these are officially supported by the development team of ETCARS and are maintained at the sole discretion of their authors.

- JavaScript: https://www.npmjs.com/package/etcars-node-client
- C#: https://github.com/shardick/etcars-dotnet-sdk

CHAPTER 4

The Data

4.1 General

There are several events emitted with several different data structures.

Here you can look and see the different data events and the different data structures included.

All data broadcast is in a JSON format with 1 exception. The string broadcast starts with an 8 character header that represents the length of the string broadcast.

For most languages, this can be ignored and all data can be received then parsed by using the first `{` and the broadcast ends with `\r`.

Using those points as a basis, a substring of the data can be made and then the JSON parsed into a proper JSON object.

Many languages have implementations for this. Otherwise, you may look at the Third Party list of packages on the connecting page.

All events have data as the root element and status available as part of the data object.

4.2 SDK Events

Not all raw sdk events are broadcasted from the server. This may change in the future.

The SCS SDK Events available are:

- SCS Telemetry Init
- SCS Telemetry Shutdown
- SCS Telemetry Configuration
- SCS Telemetry Frame Start
- SCS Telemetry Frame End
- SCS Telemetry Pause

4.3 Events

Some events share the same data structure, and it's the first one documented below. Others are special events that have a different structure.
The normal telemetry structure is shared by the following events:

- Intialized
- Telemetry
- Job Started
- Job Continue
- Job Finished
- Paused
- UnPaused
- No Lights
- Telemetry Shut Down

The special events are as follows:

- Speeding
- Collision
- Late

4.3.1 Initialization Event

This event is fired when the game is first fired up and initialization has been completed. Do not expect to capture this event as this is also when the server is being started. Unless you connect exactly when the game is started and the user clicks OK on the Additional Features screen, this event may not be received. The status contained in the data object for this event is `INTIALIZED`. Available since version: 0.13

4.3.2 Telemetry Event

This event is fired every second to all connected clients. The status contained in the data object for this event is `TELEMETRY`. Available since version: 0.13

4.3.3 Job Started Event

This event is fired when a job has been started. While a job may be chosen in the menu, this event does not fire until the trailer has been connected. The status contained in the data object for this event is `JOB STARTED`. Available since version: 0.13

4.3.4 Job Continue Event

This event is fired when a job is being resumed. When the game is exited cleanly, the current delivery information is saved to a file for the next time the delivery is resumed. The status contained in the data object for this event is `JOB CONTINUE`. Available since version: 0.14

4.3.5 Job Finished Event

This event is fired when a job has been completed. This is fired based off of the in-game configuration event and is fired AFTER the user clicks continue on the job results screen. The status contained in the data object for this event is `JOB FINISH`. Available since version: 0.13

This event is fired when the game has been paused. The status contained in the data object for this event is `PAUSED`. Available since version: 0.13

4.3.6 Game UnPaused Event

This event is fired when the game is unpaused. The status contained in the data object for this event is `UNPAUSED`. Available since version: 0.13

4.3.7 No Lights Event

This event is fired when drivers should have their lights on but do not. This uses game time to calculate 7AM to 7PM. The status contained in the data object for this event is `NO LIGHTS`. Available since version: 0.13

4.3.8 Telemetry Shut Down Event

This event is fired when the game is closing. This event may not be captured due to the way the SDK works. The status contained in the data object for this event is `TELEMETRY SHUT DOWN`. Available since version: 0.13

4.3.9 Collision Event

This event is fired when the minimal detectable damage has been detected, this results in a possible collision. The player may have collided with a pole, AI, another driver, etc… The status contained in the data object for this event is `COLLISION`. Available since version: 0.13

4.3.10 Late Delivery Event

This event is fired when the delivery is now past the time due for the game. The status contained in the data object for this event is `LATE`. Available since version: 0.13

4.3.11 Speeding Event

This event is fired when the truck speed exceeds the speed limit by 7 meters / second(~15mph/~25kmh) The status contained in the data object for this event is `SPEEDING`. Available since version: 0.13

4.4 Structures

4.4.1 Normal Telemetry Structure

For more detailed information, consult SCS Software's SDK and the examples it contains. Especially for Velocities, Accelerations, and Offsets. The normal telemetry structure is as follows:

```
########{
    data:<Object>
    {
        status: <string>,
        telemetry: <Object>
        {
            pluginVersion:<Object>
            {
                majorVersion:  <int>,
                minorVersion: <int>,
                pluginVersionOnlyStr: <string>
            },
            game:<Object>
            {
                isMultiplayer: <boolean>,
                paused: <boolean>,
                isDriving: <boolean>,
                majorVersion: <int>,
                minorVersion: <int>,
                gameID: <string>, value(s): "ets2","ats"
                gameName: <string>, value(s): "Euro Truck Simulator 2", "American␣
↪Truck Simulator"
                gameVersionStr: <string>,
                gameVersionOnly: <string>,
                nextRestStop: <int>, In-game minutes until next rest stop. This value␣
↪may be provided regardless if fatigue simulation is enabled or disabled.
                gameDateTime: <unsigned int>, Number of in-game minutes elapsed since␣
↪Jan 1, 2001.
                gameDayTime: <string>, Formatted string representing the in-game day␣
↪and time. Example: Mon 14:53
                gameDateTimeStr: <string>, Formatted string representing the in-game␣
↪date and time. Example: 2001-01-01 14:53
                osEnvironment: <string>, The current operating system being used.␣
↪value(s): "Linux", "Windows"
                architecture: <string>, The architecture currently running. value(s):
↪"x86","x64"
                localScale: <float>, The scale applied to distance and time to␣
↪compensate for the scale of the map. 1s real time = localScale time. Example:␣
↪localScale is 19, in 1 second real time, 19 seconds passed in-game. NOTE: 1:1 maps␣
↪will not provide this channel.
                substances: <array<string>>, Array of strings representing the␣
↪configured substances. This is used for the wheel substance to determine the␣
↪substance each wheel is touching.
            },
            "truck":
            {
                cruiseControlSpeed: <float>, Speed selected for cruise control. 0 if␣
↪disabled.
                gear: <int>, Gear currently selected in the engine. >0 - Forward␣
↪gears, 0 - neutral, <0 - reverse gears.
```

(continues on next page)

(continued from previous page)

```
                gearDisplayed: <int>, same as gear, except this is the gear displayed␣
↪in the dashboard.
                retarderBrakeLevel: <unsigned int>, current level of the retarder. <0,
↪max> where 0 is disabled retarder and max is the maximal value in the truck␣
↪configuration.
                wipersOn: <boolean>, wipers on the truck on or off.
                make: <string>, The make of the current truck in the in-game language.
↪ Encoded using utf8mb4.
                makeID: <string>, The make of the current truck, limited to C-
↪identifier characters and dots.
                model: <string>, The model of the current truck in the in-game␣
↪language. Encoded using utf8mb4.
                modelID: <string>, The model of the current truck, limted to C-
↪identifier characters and dots.
                shifterType: <string>, The type of shifter the controls are␣
↪configured for.
                odometer: <float>, The odometer of the truck, represented in␣
↪kilometers.
                hasTruck: <boolean>, Whether or not the configured truck is the␣
↪selected truck in game. If false, the truck information is no longer valid and the␣
↪user has no truck.
                engineEnabled: <boolean>, Whether or not the engine is enabled on the␣
↪configured truck.
                electricsEnabled: <boolean>, Whether or not the electronics are␣
↪enabled on the configured truck.
                motorBrake: <boolean>, Whether or not the motor brake is enabled/
↪engaged.
                parkingBrake: <boolean>, Whether or not the parking brake is enabled/
↪engaged.
                speed: <float>, The speed of the truck in meters per second.
                engineRPM: <float>, The number of RPMs the engine is outputting.
                brakeTemperature: <float>, The loose approximation of the temperature␣
↪of the brakes in degrees celsius. This is for the entire truck and not at the wheel␣
↪level.
                fuelRange: <float>, Estimated range of the current truck with current␣
↪amount of fuel in km.
                oilPressure: <float>, Pressure of the oil in psi(pounds per square␣
↪inch).
                oilTemperature: <float>, The loose approximation of the temperature␣
↪of the oil in degrees celsius.
                waterTemperature: <float>, The loose approximation of the temperature␣
↪of the water in degrees celsius.
                batteryVoltage: <float>, The loose approximation of the voltage of␣
↪the battery.
                inputSteering: <float>, Steering received from input. <-1,1>, Right =␣
↪-1.0.
                inputThrottle: <float>, Throttle received from input. <0,1>.
                inputBrake: <float>, Brake received from input. <0,1>.
                inputClutch: <float>, Clutch received from input. <0,1>.
                effectiveSteering: <float>, Steering as used by the simulation,␣
↪accounts for interpolation speeds and simulated counterforces for inputs. <-1,1>.
                effectiveThrottle: <float>, Throttle as used by the simulation.␣
↪Accounts for the press attack curve for inputs and cruise control input. <0,1>.
                effectiveBrake: <float>, Brake as used by the simulation. Accounts␣
↪for the press attack curve for inputs, excludes retarder, parking and motor brakes.
↪<0,1>.
                effectiveClutch: <float>, Clutch input as used by the simulation.␣
↪Accounts for automatic shifting/interpolation of player input. <0,1>.
```

(continues on next page)

(continued from previous page)

```
                hShifterSlot: <unsigned int>, Gearbox slot the h-shifter handle is␣
↪currently in, 0 = no slot selected.
                brakeAirPressure: <float>, Pressure in the brake air tank in␣
↪psi(pounds per square inch).
                adBlue: <float>, Amount of adBlue in litres. NOTE: This value will␣
↪always be 0 in American Truck Simulator(ats).
                daasboardBacklight: <float>, Intensity of the dashboard backlight as␣
↪factor. <0,1>.
                forwardGearCount: <unsigned int>, Number of forward gears on␣
↪undamaged truck.
                reverseGearCount: <unsigned int>, Number of reverse gears on␣
↪undamaged truck.
                retarderStepCount: <unsigned int>, Number of steps in the retarder.␣
↪Zero if retarder is not mounted to truck.
                trailerConnected: <boolean>, Whether or not there is a trailer␣
↪connected to the truck.
                worldPlacement: <Object>
                {
                    x: <float>, X Coordinate of the truck.
                    y: <float>, Y Coordinate of the truck.
                    z: <float>, Z Coordinate of the truck.
                    heading: <float>, Heading of the truck. <0,0.9999> 0/1 = north, 0.
↪75 = east, 0.5 = south, 0.25 = west
                    pitch: <float>, Pitch of the truck.
                    roll: <float>, Roll of the truck.
                },
                linearVelocity: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                angularVelocity: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                linearAcceleration: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                angularAcceleration: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                cabinOffset: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                    heading: <float>,
                    pitch: <float>,
```

(continues on next page)

(continued from previous page)

```
                        roll: <float>,
                    },
                    hookPosition: <Object>
                    {
                        x: <float>,
                        y: <float>,
                        z: <float>,
                    },
                    headPosition: <Object>
                    {
                        x: <float>,
                        y: <float>,
                        z: <float>,
                    },
                    cabinAngularVelocity: <Object>
                    {
                        x: <float>,
                        y: <float>,
                        z: <float>,
                    },
                    cabinAngularAcceleration: <Object>
                    {
                        x: <float>,
                        y: <float>,
                        z: <float>,
                    },
                    forwardRatios: <array<float>>, Array of floats representing each␣
↪forward gear ratio.
                    reverseRatios: <array<float>>, Array of floats representing each␣
↪reverse gear ratio.
                    wheelCount: <int>, Number of simulated wheels on the current truck.
                    wheelInfo: <array<Object>>, Array of wheel information, the length of␣
↪this array should match the wheel count. Each wheel object will have the following␣
↪properties:
                    [
                        {
                            suspensionDeflection: <float>, Vertical displacement of the␣
↪wheel from its axis in meters.
                            onGround: <boolean>, Wheter or not the wheel is on the ground.
                            substance: <string>, The current substance the wheel is␣
↪touching. This is value is in the substances array. Default/Air: static.
                            angularVelocty: <float>, Angular velocity of the wheel in␣
↪rotations per second. Positive = forward.
                            lift: <float>, Lift state of the wheel. <0,1>. 0 = non-lifted,
↪ 1 = fully lifted, zero for non-liftable axles.
                            liftOffset: <float>, Vertical displacement of the wheel axle␣
↪from its normal position in meters as result of lifting. Might have non-linear␣
↪relation to lift ratio. Set to zero for non-liftable axles.
                            position: <Object>
                            {
                                x: <float>,
                                y: <float>,
                                z: <float>,
                            },
                            steerable: <boolean>, Whether or not this wheel is steerable.
                            simulated: <boolean>, Whether or not this wheel is physically␣
↪simulated.
```

(continues on next page)

(continued from previous page)

```
                        radius: <float>, Radius of the wheel. in meters??????
                        steering: <float>, Steering rotation of the wheel in␣
↪rotations. <-0.25,0.25>. Counterclockwise direction when looking from top. 0.25 =␣
↪left,  -0.25 = right. Set to zero for non-steered wheels.
                        rotation: <float>, Rolling rotation of the wheel in rotations.
↪ <0,1> range in which value increase corresponds to forward movement.
                        powered: <boolean>, Whether or not this wheel is powered.
                        liftable: <boolean>, Whether or not this wheel is liftable.
                    }
                ],
                trailerPlacement: <Object> Trailer placement, similar to word␣
↪placement.
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                    heading: <float>,
                    pitch: <float>,
                    roll: <float>
                },
                warnings: <Object>
                {
                    batteryVoltage: <boolean>, Whether or not the battery voltage␣
↪warning light is on.
                    airPressure: <boolean>, Whether or not the air pressure warning␣
↪light is on.
                    airPressureEmergency: <boolean>, Whether or not the air pressure␣
↪emergency warning light is on.
                    oilPressure: <boolean>, Whether or not the oil pressure warning␣
↪light is on.
                    waterTemperature: <boolean>, Whether or not the water temperature␣
↪warning light is on.
                    fuelLow: <boolean>, Whether or not the fuel warning light is on.
                    adBlue: <boolean>, Whether or not the adBlue warning light is on.␣
↪NOTE: Always false in American Truck Simulator(ats).
                },
                damages: <Object> All damages are floats with a range of <0,1> where␣
↪1 is 100% damage and 0 is 0% damage.
                {
                    engine: <float>,
                    transmission: <float>,
                    cabin: <float>,
                    chassis: <float>,
                    wheels: <float>,
                    trailer: <float>, NOTE: 0 if no trailer is configured.
                },
                lights:
                {
                    lowBeam: <boolean>, Whether or not the low beams are on or off.
                    highBeam:  <boolean>, Whether or not the high beams are on or off.
                    frontAux: <int>, Front auxilary light status. 0 = off, 1 = dimmed,
↪ 2 = full brightness.
                    beacon:  <boolean>, Whether or not the beacon(s) is(are) on or␣
↪off.
                    parking:  <boolean>, Whether or not the parking lights are on or␣
↪off.
                    brake:  <boolean>, Whether or not the brake lights are on or off.
```

(continues on next page)

(continued from previous page)

```
                    reverse:  <boolean>, Whether or not the reverse lights are on or␣
↪off.
                    leftBlinkerEnabled: <boolean>, Whether or not the left blinker␣
↪lights are activated or not.
                    rightBlinkerEnabled: <boolean>, Whether or not the right blinker␣
↪lights are activated or not.
                    leftBlinkerOn:  <boolean>, Whether or not the left blinker lights␣
↪are on or off.
                    rightBlinkerOn:  <boolean>, Whether or not the right blinker␣
↪lights are on or off.
                    roofAux: <int>, Roof auxilary light status. 0 = off, 1 = dimmed,␣
↪2 = full brightness.
                },
                fuel:
                {
                    capacity: <float>, Amount of fuel the tank can hold in litres.
                    warningLevel: <float>, Amount of fuel remaining in the tank at␣
↪which the warning light will turn on. <0,1>. To calculate the litres, simply take␣
↪capacity and multiply it by this value. Example: capacity of 100 litres *␣
↪warningLevel of 0.15 = 15 Litres. At 15 litres, the fuel warning light will turn on␣
↪in this example.
                    consumptionAverage: <float>, Average consumption of the fuel in␣
↪litres/km.
                    currentLitres: <float>, Amount of fuel currently in the tank.
                }
            },
            navigation: <Object>
            {
                distance: <float>, Navigation distance remaining until the next␣
↪waypoint is hit OR distance remaining until the company of the delivery is reached.␣
↪Represented in meters. NOTE: When electronics are disabled or when route advisor is␣
↪completely disabled, this value will ALWAYS be 0.
                time: <float>, Estimated number of in-game minutes until the next␣
↪destination point is reached.
                lowestDistance: <float>, The lowest distance reported.
                speedLimit: <float>, The current speed limit in meters per second. 0␣
↪= no speed limit.
            },
            job: <Object>
            {
                cargoID: <string>, ID of the cargo being transported. Limited to C-
↪identifier characters and dots.
                cargo: <string>, Name of the cargo being transported in the in-game␣
↪language. Encoded in utf8mb4.
                mass: <float>, Mass of the cargo in kilograms.
                income: <float>, Total expected income of the delivery in USD for␣
↪American Truck Simulator(ats). EUR for Euro Truck Simulator 2(ets2). This value␣
↪will not update due to damage, bonuses, or on the finish screen.
                destinationCityID: <string>, ID of the destination city. Limited to C-
↪identifier characters and dots.
                destinationCity: <string>, Name of the destination city in the in-
↪game language. Encoded in utf8mb4.
                destinationCompanyID: <string>, ID of the destination company.␣
↪Limited to C-identifier characters and dots.
                destinationCompany: <string>, Name of the destination company in the␣
↪in-game language. Encoded in utf8mb4.
                sourceCityID: <string>, ID of the source city. Limited to C-
↪identifier characters and dots.
```

(continues on next page)

(continued from previous page)

```
                sourceCity: <string>, Name of the source city in the in-game language.
↪ Encoded in utf8mb4.
                sourceCompanyID: <string> ID of the source company. Limited to C-
↪identifier characters and dots.
                sourceCompany: <string> Name of the source company in the in-game␣
↪language. Encoded in utf8mb4.
                deliveryTime: <int> The time at which the delivery will be considered␣
↪late in-game. Represented in number of minutes since Jan 1, 2001.
                isLate: <boolean>, Whether or not the delivery is considered late.
                timeRemaining: <int>, Number of in-game minutes remaining until the␣
↪delivery is late.
            },
            trailer: <Object>
            {
                linearVelocity: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                angularVelocity: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                linearAcceleration: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                angularAcceleration: <Object>
                {
                    x: <float>,
                    y: <float>,
                    z: <float>,
                },
                wheelCount: <int>, Number of wheels on the trailer.
                wheelInfo: <array<Object>>, Array of objects that represent the wheel␣
↪information. The number of objects in this array should match the wheelCount. The␣
↪object layout is shown below:
                [
                    {
                        suspensionDeflection: <float>, Vertical displacement of the␣
↪wheel from its axis in meters.
                        onGround: <boolean>, Wheter or not the wheel is on the ground.
                        substance: <string>, The current substance the wheel is␣
↪touching. This is value is in the substances array. Default/Air: static.
                        angularVelocty: <float>, Angular velocity of the wheel in␣
↪rotations per second. Positive = forward.
                        lift: <float>, Lift state of the wheel. <0,1>. 0 = non-lifted,
↪ 1 = fully lifted, zero for non-liftable axles.
                        liftOffset: <float>, Vertical displacement of the wheel axle␣
↪from its normal position in meters as result of lifting. Might have non-linear␣
↪relation to lift ratio. Set to zero for non-liftable axles.
                        position: <Object>
```

(continues on next page)

(continued from previous page)

```
                        {
                            x: <float>,
                            y: <float>,
                            z: <float>,
                        },
                        steerable: <boolean>, Whether or not this wheel is steerable.
                        simulated: <boolean>, Whether or not this wheel is physically␣
↪simulated.
                        radius: <float>, Radius of the wheel. in meters??????
                        steering: <float>, Steering rotation of the wheel in␣
↪rotations. <-0.25,0.25>. Counterclockwise direction when looking from top. 0.25 =␣
↪left,  -0.25 = right. Set to zero for non-steered wheels.
                        rotation: <float>, Rolling rotation of the wheel in rotations.
↪ <0,1> range in which value increase corresponds to forward movement.
                        powered: <boolean>, Whether or not this wheel is powered.
                        liftable: <boolean>, Whether or not this wheel is liftable.
                    }
                ],
                id: <string>, The trailer id. Limited to C-identifier characters and␣
↪dots.
                cargoAccessoryId: <string>, The id of the accessory attatched to the␣
↪trailer. Limited to C-identifier characters and dots.
                hookPosition: <Object> Position of the trailer connecttion hook in␣
↪vehicle space
                {
                    x: <float>,
                    y: <float>,
                    z: <float>
                }
            },
            user: <Object> Information about the current user. This data is gathered␣
↪using the Steamworks SDK.
            {
                steamID: <string>, The user's steamID64(SteamID).
                steamUsername: <string>, The user's profile name.
                DLC: <Object>
                {
                    DLC: <array<Object>> An array representing the game DLCs. This␣
↪does NOT include mods or mods downloaded from the steam workshop. The layout of the␣
↪object is shown below:
                    [
                        {
                            appid: <int>, The steam appid. Can be used with the steam␣
↪web api.
                            name: <string>, The name of the DLC according to steam.
                            available: <boolean>, Whether or not this DLC is␣
↪available for purchase.
                            installed: <boolean>, Whether or not this user has the␣
↪DLC installed.
                        }
                    ]
                }

            }
        },
        jobData: <Object> This is a representation of what some collected job data␣
↪would be for a virtual trucking company or reporting service. Cirtical parts of␣
↪this data are saved to encrypted files for job resume, however it is possible for␣
↪users to edit the data if it is decrypted properly. USE AT YOUR OWN RISK. All␣
↪damages are floats with a range of <0,1> where 1 is 100% damage and 0 is 0% damage.
```

(continues on next page)

(continued from previous page)

```
        {
            status: <int>, 1 = In progress, 2 = Finished, 3 = Cancelled NOTE:␣
↪Cancellation detection is difficult and may not be functioning properly.
            wasSpeeding: <boolean>, Whether or not the user was speeding on the␣
↪delivery. Speeding qualifies as 7 meters per second ABOVE the reported speed limit.
            jobStartedEventFired: <boolean>, Whether or not the job started event has␣
↪been fired.
            isMultiplayer: <boolean>, Whether or not the user is in TruckersMP.
            late: <boolean>, Whether or not the delivery is considered late.
            onJob: <boolean>, Whether or not the user is still on the delivery.
↪(False=delivery finished or cancelled).
            wasFinished: <boolean>, Whether or not the delivery was completed. May␣
↪not be accurate for certain types of deliveries.
            wasTrailerDisconnected: <boolean>, Whether or not the trailer was␣
↪disconnected at time of "finish". Used in cancellation detection.
            cargoID: <string>, ID of the cargo being transported. Limited to C-
↪identifier characters and dots.
            cargo: <string>, Name of the cargo being transported in the in-game␣
↪language. Encoded in utf8mb4.
            trailerMass: <float>, Mass of the cargo in kilograms.
            income: <float>, Total expected income of the delivery in USD for␣
↪American Truck Simulator(ats). EUR for Euro Truck Simulator 2(ets2). This value␣
↪will not update due to damage, bonuses, or on the finish screen.
            destinationCityID: <string>, ID of the destination city. Limited to C-
↪identifier characters and dots.
            destinationCity: <string>, Name of the destination city in the in-game␣
↪language. Encoded in utf8mb4.
            destinationCompanyID: <string>, ID of the destination company. Limited to␣
↪C-identifier characters and dots.
            destinationCompany: <string>, Name of the destination company in the in-
↪game language. Encoded in utf8mb4.
            sourceCityID: <string>, ID of the source city. Limited to C-identifier␣
↪characters and dots.
            sourceCity: <string>, Name of the source city in the in-game language.␣
↪Encoded in utf8mb4.
            sourceCompanyID: <string> ID of the source company. Limited to C-
↪identifier characters and dots.
            sourceCompany: <string> Name of the source company in the in-game␣
↪language. Encoded in utf8mb4.
            deliveryTime: <int> The time at which the delivery will be considered␣
↪late in-game. Represented in number of minutes since Jan 1, 2001.
            isLate: <boolean>, Whether or not the delivery is considered late.
            timeRemaining: <int>, Number of in-game minutes remaining until the␣
↪delivery is late.
            truckMake: <string>, The make of the current truck in the in-game␣
↪language. Encoded using utf8mb4.
            truckMakeID: <string>, The make of the current truck, limited to C-
↪identifier characters and dots.
            truckModel: <string>, The model of the current truck in the in-game␣
↪language. Encoded using utf8mb4.
            truckModelID: <string>, The model of the current truck, limted to C-
↪identifier characters and dots.
            gameID: <string>, value(s): "ets2","ats"
            game: <string>, value(s): "Euro Truck Simulator 2", "American Truck␣
↪Simulator"
            gameVersion: <string>,
            pluginVersion: <string>,
```

(continues on next page)

(continued from previous page)

```
            topSpeed: <float>, The top speed reported for the delivery in meters per␣
↪second.
            speedingCount: <int>, Number of speeding incidents on this delivery.
            distanceDriven: <float>, Total distance driven from the time the trailer␣
↪is picked up to the delivery finished event in km.
            fuelBurned: <float>, Total amount of fuel burned for the delivery in␣
↪litres.
            fuelPurchased: <float>, Total amount of fuel purchased while on the␣
↪delivery in litres.
            startOdometer: <float>, The starting odometer for the delivery in␣
↪kilometers.
            endOdometer: <float>, The ending odometer for the delivery in kilometers.
            timeRemaining: <int>, The number of in-game minutes remaining until the␣
↪delivery is considered late.
            timeStarted: <long long>, The in-game time the delivery was started.␣
↪Minutes since Jan 1 2001.
            timeDue: <long long>, The in-game time the delivery is due. Minutes since␣
↪Jan 1 2001.
            timeDelivered: <long long>, The in-game time the delivery was finished.␣
↪Minutes since Jan 1 2001.
            collisionCount: <int>, Total number of estimated in-game collisions that␣
↪ocurred on the job.
            finishTrailerDamage: <float>, Finishing trailer damage.
            startTrailerDamage: <float>, Starting trailer damage.
            deliveryX: <float>, X Coordinate of the truck when the delivery was␣
↪finished.
            deliveryY: <float>, Y Coordinate of the truck when the delivery was␣
↪finished.
            deliveryZ: <float>, Z Coordinate of the truck when the delivery was␣
↪finished.
            pickupX: <float>, X Coordinate of the truck when the trailer was picked␣
↪up.
            pickupY: <float>, Y Coordinate of the truck when the trailer was picked␣
↪up.
            pickupZ: <float>, Z Coordinate of the truck when the trailer was picked␣
↪up.
            trailerDeliveryX: <float>, X Coordinate of the trailer when the delivery␣
↪was finished.
            trailerDeliveryY: <float>, Y Coordinate of the trailer when the delivery␣
↪was finished.
            trailerDeliveryZ: <float>, Z Coordinate of the trailer when the delivery␣
↪was finished.
            trailerPickupX: <float>, X Coordinate of the trailer when the trailer was␣
↪picked up.
            trailerPickupY: <float>, Y Coordinate of the trailer when the trailer was␣
↪picked up.
            trailerPickupZ: <float>, Z Coordinate of the trailer when the trailer was␣
↪picked up.
            startEngineDamage: <float>, Starting engine damage.
            startTransmissionDamage: <float>, Starting transmission damage.
            startCabinDamage: <float>, Starting cabin damage.
            startChassisDamage: <float>, Starting chassis damage.
            startWheelDamage: <float>, Starting wheel damage.
            finishEngineDamage: <float>, Finishing engine damage.
            finishTransmissionDamage: <float>, Finishing transmission damage.
            finishCabinDamage: <float>, Finishing cabin damage.
            finishChassisDamage: <float>, Finishing chassis damage.
```

(continues on next page)

(continued from previous page)

```
            finishWheelDamage: <float>, Finishing wheel damage.
            totalEngineDamage: <float>, Total engine damage.
            totalTransmissionDamage: <float>, Total transmission damage.
            totalCabinDamage: <float>, Total cabin damage.
            totalChassisDamage: <float>, Total chassis damage.
            totalWheelDamage: <float>, Total wheel damage.
            totalTrailerDamage: <float>, Total trailer damage.
            navigationDistanceRemaining: <float>, Navigation distance remaining for␣
↪the delivery.
            fuel: <float>, Current fuel in the tank in litres.
            odometer: <float>, Truck odometer in km.
            engineDamage: <float>, Current engine damage.
            transmissionDamage: <float>, Current transmission damage.
            cabinDamage: <float>, Current cabin damage.
            chassisDamage: <float>, Current chassis damage.
            wheelDamage: <float>, Current wheel damage.
            trailerDamage: <float>, Current trailer damage.
            realTimeStarted: <long long>, Time since EPOCH that the delivery was␣
↪picked up.
            realTimeEnded: <long long>, Time since EPOCH that the delivery was␣
↪finished.
            realTimeTaken: <long long>, Real time milliseconds spent on delivery.
        }
    }
}\r
```

4.4.2 Speeding Structure

The speeding structure is as follows:

```
########{
    data: <Object>
    {
        status: <string>, value: SPEEDING
        offense: <Object>
        {
            reason: <string>, value: speeding
            speed: <float>, truck speed in meters / second
            speedLimit: <float>, speed limit in meters / second, NOTE: 0 means no␣
↪speed limit
            x: <float>,
            y: <float>,
            z: <float>,
            trailerX: <float>, NOTE: value is 0.00 with no trailer
            trailerY: <float>, NOTE: value is 0.00 with no trailer
            trailerZ: <float>, NOTE: value is 0.00 with no trailer
        }
    }
}\r
```

4.4.3 Collision Structure

The collision structure is as follows:

```
########{
    data: <Object>
    {
        status: <string>, value: COLLISION
        offsense: <Object>
        {
            reason: <string>, value: collision
            speed: <float>, NOTE: truck speed in meters / second
            speedLimit: <float>, NOTE: speed limit in meters / second
            x: <float>,
            y: <float>,
            z: <float>,
            trailerX: <float>, NOTE: value is 0.00 with no trailer
            trailerY: <float>, NOTE: value is 0.00 with no trailer
            trailerZ: <float>, NOTE: value is 0.00 with no trailer
            cabinDamage: <float>, <0,1>
            chassisDamage: <float>, <0,1>
            engineDamage: <float>, <0,1>
            transmissionDamage: <float>, <0,1>
            trailerDamage: <float>, <0,1>, NOTE: value is 0.00 with no trailer
        }
    }
}\r
```

4.4.4 Late Structure

The late structure is as follows:

```
########{
    data: <Object>
    {
        status: <string>, value: LATE,
        offense: <Object>
        {
            reason: <string>, value: late
            timeDue: <unsigned int>, This number represents the game time that this␣
↪delivery will be consiedered late. Represented in number of minutes since Jan 1,␣
↪2001.
            currentGameTime: <unsigned int>, This number represents the number of␣
↪minutes elapsed in-game since Jan 1, 2001.
            x: <float>,
            y: <float>,
```

(continues on next page)

(continued from previous page)

```
            z: <float>,
            trailerX: <float>, NOTE: value is 0.00 with no trailer
            trailerY: <float>, NOTE: value is 0.00 with no trailer
            trailerZ: <float>, NOTE: value is 0.00 with no trailer
            fee: <float>, NOTE: not provided by game, just a simple estimation or␣
↪suggested value in the game's default currency.
        }
    }
}\r
```