**OVERVIEW**

The F1 series of games support the outputting of key game data via a UDP data stream. This data can be interpreted by external apps or connected peripherals for a range of different uses, including providing additional telemetry information, customised HUD displays, motion platform hardware support or providing force feedback data for custom steering wheels. The following information is a summary of the data that is outputted so that developers of supporting hardware or software are able to configure these to work with the F1 game correctly. If the information you require is not contained here, or if you have any issues with the UDP data itself, then please let us know and a member of the dev team will respond to your query as soon as possible.

**PACKET TYPES**

 The main change for 2018 is the introduction of multiple packet types: each packet can now carry different types of data rather than having one packet which contains everything. A header has been added to each packet as well so that versioning can be tracked and it will be easier for applications to check they are interpreting the incoming data in the correct way.

Each packet has the following header:

struct PacketHeader

{

uint16 m\_packetFormat; // 2018

uint8 m\_packetVersion; // Version of this packet type, all start from 1

uint8 m\_packetId; // Identifier for the packet type, see below

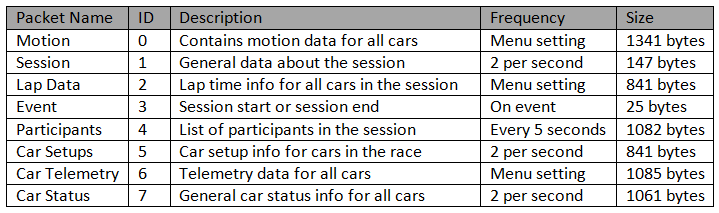
uint64 m\_sessionUID; // Unique identifier for the session

float m\_sessionTime; // Session timestamp

uint m\_frameIdentifier; // Identifier for the frame the data was retrieved on

uint8     m\_playerCarIndex;       // Index of player's car in the array

};

[](https://us.v-cdn.net/5021484/uploads/editor/i2/fj958zeqdhf8.png)

**MOTION PACKET**The motion packet gives physics data for all the cars being driven. There is additional data for the car being driven with the goal of being able to drive a motion platform setup.

***N.B.*** For the normalised vectors below, to convert to float values divide by 32767.0f. 16-bit signed values are used to pack the data and on the assumption that direction values are always between -1.0f and 1.0f.  
Frequency: Rate as specified in menus  
Size: 1341 bytes

struct CarMotionData

{

float m\_worldPositionX; // World space X position

float m\_worldPositionY; // World space Y position

float m\_worldPositionZ; // World space Z position

float m\_worldVelocityX; // Velocity in world space X

float m\_worldVelocityY; // Velocity in world space Y

float m\_worldVelocityZ; // Velocity in world space Z

int16 m\_worldForwardDirX; // World space forward X direction (normalised)

int16 m\_worldForwardDirY; // World space forward Y direction (normalised)

int16 m\_worldForwardDirZ; // World space forward Z direction (normalised)

int16 m\_worldRightDirX; // World space right X direction (normalised)

int16 m\_worldRightDirY; // World space right Y direction (normalised)

int16 m\_worldRightDirZ; // World space right Z direction (normalised)

float m\_gForceLateral; // Lateral G-Force component

float m\_gForceLongitudinal; // Longitudinal G-Force component

float m\_gForceVertical; // Vertical G-Force component

float m\_yaw; // Yaw angle in radians

float m\_pitch; // Pitch angle in radians

float m\_roll; // Roll angle in radians

};

struct PacketMotionData

{

PacketHeader m\_header; // Header

CarMotionData m\_carMotionData[20]; // Data for all cars on track

// Extra player car ONLY data

float m\_suspensionPosition[4]; // Note: All wheel arrays have the following order:

float m\_suspensionVelocity[4]; // RL, RR, FL, FR

float m\_suspensionAcceleration[4]; // RL, RR, FL, FR

float m\_wheelSpeed[4]; // Speed of each wheel

float m\_wheelSlip[4]; // Slip ratio for each wheel (added v1.2.1)

float m\_localVelocityX; // Velocity in local space

float m\_localVelocityY; // Velocity in local space

float m\_localVelocityZ; // Velocity in local space

float m\_angularVelocityX; // Angular velocity x-component

float m\_angularVelocityY; // Angular velocity y-component

float m\_angularVelocityZ; // Angular velocity z-component

float m\_angularAccelerationX; // Angular velocity x-component

float m\_angularAccelerationY; // Angular velocity y-component

float m\_angularAccelerationZ; // Angular velocity z-component

float m\_frontWheelsAngle; // Current front wheels angle in radians (added v1.2.1)

};

**SESSION PACKET**

The session packet includes details about the current session in progress.

Frequency: 2 per second

Size: 147 bytes (updated v1.3)

struct MarshalZone

{

float m\_zoneStart; // Fraction (0..1) of way through the lap the marshal zone starts

int8 m\_zoneFlag; // -1 = invalid/unknown, 0 = none, 1 = green, 2 = blue, 3 = yellow, 4 = red

};

struct PacketSessionData

{

PacketHeader m\_header; // Header

uint8 m\_weather; // Weather - 0 = clear, 1 = light cloud, 2 = overcast

// 3 = light rain, 4 = heavy rain, 5 = storm

int8 m\_trackTemperature; // Track temp. in degrees celsius

int8 m\_airTemperature; // Air temp. in degrees celsius

uint8 m\_totalLaps; // Total number of laps in this race

uint16 m\_trackLength; // Track length in metres

uint8 m\_sessionType; // 0 = unknown, 1 = P1, 2 = P2, 3 = P3, 4 = Short P

// 5 = Q1, 6 = Q2, 7 = Q3, 8 = Short Q, 9 = OSQ

// 10 = R, 11 = R2, 12 = Time Trial

int8 m\_trackId; // -1 for unknown, 0-21 for tracks, see appendix

uint8 m\_era; // Era, 0 = modern, 1 = classic

uint16 m\_sessionTimeLeft; // Time left in session in seconds

uint16 m\_sessionDuration; // Session duration in seconds

uint8 m\_pitSpeedLimit; // Pit speed limit in kilometres per hour

uint8 m\_gamePaused; // Whether the game is paused

uint8 m\_isSpectating; // Whether the player is spectating

uint8 m\_spectatorCarIndex; // Index of the car being spectated

uint8 m\_sliProNativeSupport; // SLI Pro support, 0 = inactive, 1 = active

uint8 m\_numMarshalZones; // Number of marshal zones to follow

MarshalZone m\_marshalZones[21]; // List of marshal zones – max 21

uint8 m\_safetyCarStatus; // 0 = no safety car, 1 = full safety car

// 2 = virtual safety car (added v1.3)

unint8 m\_networkGame; // 0 = offline, 1 = online (added v1.3)

};

**LAP DATA PACKET**

The lap data packet gives details of all the cars in the session.

Frequency: Rate as specified in menus

Size: 841 bytes (updated v1.3)

struct LapData

{

float m\_lastLapTime; // Last lap time in seconds

float m\_currentLapTime; // Current time around the lap in seconds

float m\_bestLapTime; // Best lap time of the session in seconds

float m\_sector1Time; // Sector 1 time in seconds

float m\_sector2Time; // Sector 2 time in seconds

float m\_lapDistance; // Distance vehicle is around current lap in metres – could

// be negative if line hasn’t been crossed yet

float m\_totalDistance; // Total distance travelled in session in metres – could

// be negative if line hasn’t been crossed yet

float       m\_safetyCarDelta;        // Delta in seconds for safety car (added v1.3)

uint8 m\_carPosition; // Car race position

uint8 m\_currentLapNum; // Current lap number

uint8 m\_pitStatus; // 0 = none, 1 = pitting, 2 = in pit area

uint8 m\_sector; // 0 = sector1, 1 = sector2, 2 = sector3

uint8 m\_currentLapInvalid; // Current lap invalid - 0 = valid, 1 = invalid

uint8 m\_penalties; // Accumulated time penalties in seconds to be added

uint8 m\_gridPosition; // Grid position the vehicle started the race in

uint8 m\_driverStatus; // Status of driver - 0 = in garage, 1 = flying lap

// 2 = in lap, 3 = out lap, 4 = on track

uint8 m\_resultStatus; // Result status - 0 = invalid, 1 = inactive, 2 = active

// 3 = finished, 4 = disqualified, 5 = not classified

// 6 = retired

};

struct PacketLapData

{

PacketHeader m\_header; // Header

LapData m\_lapData[20]; // Lap data for all cars on track

};

**EVENT PACKET**

This packet gives details of events that happen during the course of the race.

Frequency: When the event occurs

Size: 25 bytes (updated v1.3)

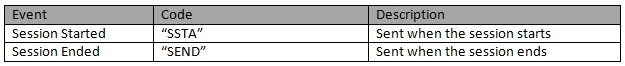
struct PacketEventData

{

PacketHeader m\_header; // Header

uint8 m\_eventStringCode[4]; // Event string code, see above

};

[](https://us.v-cdn.net/5021484/uploads/editor/3p/n40iwrzvzhwq.jpg)

**PARTICIPANTS PACKET**

This is a list of participants in the race. If the vehicle is controlled by AI, then the name will be the driver name. If this is a multiplayer game, the names will be the Steam Id on PC, or the LAN name if appropriate. On Xbox One, the names will always be the driver name, on PS4 the name will be the LAN name if playing a LAN game, otherwise it will be the driver name.

Frequency: Every 5 seconds

Size: 1082 bytes (updated v1.3)

struct ParticipantData

{

uint8 m\_aiControlled; // Whether the vehicle is AI (1) or Human (0) controlled

uint8 m\_driverId; // Driver id - see appendix

uint8 m\_teamId; // Team id - see appendix

uint8      m\_raceNumber;             // Race number of the car (added v1.3)

uint8      m\_nationality;            // Nationality of the driver (added v1.3)

char m\_name[48]; // Name of participant in UTF-8 format – null terminated

// Will be truncated with … (U+2026) if too long

};

struct PacketParticipantsData

{

PacketHeader m\_header; // Header

uint8 m\_numCars; // Number of cars in the data

ParticipantData m\_participants[20];

// m\_playerCarIndex removed in v1.3

};

**CAR SETUPS PACKET**

This packet details the car setups for each vehicle in the session. Note that in multiplayer games, other player cars will appear as blank, you will only be able to see your car setup and AI cars.

Frequency: Every 5 seconds

Size: 841 bytes (updated in v1.3)

struct CarSetupData

{

uint8 m\_frontWing; // Front wing aero

uint8 m\_rearWing; // Rear wing aero

uint8 m\_onThrottle; // Differential adjustment on throttle (percentage)

uint8 m\_offThrottle; // Differential adjustment off throttle (percentage)

float m\_frontCamber; // Front camber angle (suspension geometry)

float m\_rearCamber; // Rear camber angle (suspension geometry)

float m\_frontToe; // Front toe angle (suspension geometry)

float m\_rearToe; // Rear toe angle (suspension geometry)

uint8 m\_frontSuspension; // Front suspension

uint8 m\_rearSuspension; // Rear suspension

uint8 m\_frontAntiRollBar; // Front anti-roll bar

uint8 m\_rearAntiRollBar; // Front anti-roll bar

uint8 m\_frontSuspensionHeight; // Front ride height

uint8 m\_rearSuspensionHeight; // Rear ride height

uint8 m\_brakePressure; // Brake pressure (percentage)

uint8 m\_brakeBias; // Brake bias (percentage)

float m\_frontTyrePressure; // Front tyre pressure (PSI)

float m\_rearTyrePressure; // Rear tyre pressure (PSI)

uint8 m\_ballast; // Ballast

float m\_fuelLoad; // Fuel load

};

struct PacketCarSetupData

{

PacketHeader m\_header; // Header

CarSetupData m\_carSetups[20];

};

**CAR TELEMETRY PACKET**

This packet details telemetry for all the cars in the race. It details various values that would be recorded on the car such as speed, throttle application, DRS etc.

Frequency: Rate as specified in menus

Size: 1085 bytes (updated in v1.3)

struct CarTelemetryData

{

uint16 m\_speed; // Speed of car in kilometres per hour

uint8 m\_throttle; // Amount of throttle applied (0 to 100)

int8 m\_steer; // Steering (-100 (full lock left) to 100 (full lock right))

uint8 m\_brake; // Amount of brake applied (0 to 100)

uint8 m\_clutch; // Amount of clutch applied (0 to 100)

int8 m\_gear; // Gear selected (1-8, N=0, R=-1)

uint16 m\_engineRPM; // Engine RPM

uint8 m\_drs; // 0 = off, 1 = on

uint8 m\_revLightsPercent; // Rev lights indicator (percentage)

uint16 m\_brakesTemperature[4]; // Brakes temperature (celsius)

uint16 m\_tyresSurfaceTemperature[4]; // Tyres surface temperature (celsius)

uint16 m\_tyresInnerTemperature[4]; // Tyres inner temperature (celsius)

uint16 m\_engineTemperature; // Engine temperature (celsius)

float m\_tyresPressure[4]; // Tyres pressure (PSI)

};

struct PacketCarTelemetryData

{

PacketHeader m\_header; // Header

CarTelemetryData m\_carTelemetryData[20];

uint32              m\_buttonStatus;        // Bit flags specifying which buttons are being

// pressed currently - see appendices (added in v1.3)

};

**CAR STATUS PACKET**

This packet details car statuses for all the cars in the race. It includes values such as the damage readings on the car.

Frequency: 2 per second

Size: 1061 bytes (updated v1.3)

struct CarStatusData

{

uint8 m\_tractionControl; // 0 (off) - 2 (high)

uint8 m\_antiLockBrakes; // 0 (off) - 1 (on)

uint8 m\_fuelMix; // Fuel mix - 0 = lean, 1 = standard, 2 = rich, 3 = max

uint8 m\_frontBrakeBias; // Front brake bias (percentage)

uint8 m\_pitLimiterStatus; // Pit limiter status - 0 = off, 1 = on

float m\_fuelInTank; // Current fuel mass

float m\_fuelCapacity; // Fuel capacity

uint16 m\_maxRPM; // Cars max RPM, point of rev limiter

uint16 m\_idleRPM; // Cars idle RPM

uint8 m\_maxGears; // Maximum number of gears

uint8 m\_drsAllowed; // 0 = not allowed, 1 = allowed, -1 = unknown

uint8 m\_tyresWear[4]; // Tyre wear percentage

uint8 m\_tyreCompound; // Modern - 0 = hyper soft, 1 = ultra soft

// 2 = super soft, 3 = soft, 4 = medium, 5 = hard

// 6 = super hard, 7 = inter, 8 = wet

// Classic - 0-6 = dry, 7-8 = wet

uint8 m\_tyresDamage[4]; // Tyre damage (percentage)

uint8 m\_frontLeftWingDamage; // Front left wing damage (percentage)

uint8 m\_frontRightWingDamage; // Front right wing damage (percentage)

uint8 m\_rearWingDamage; // Rear wing damage (percentage)

uint8 m\_engineDamage; // Engine damage (percentage)

uint8 m\_gearBoxDamage; // Gear box damage (percentage)

uint8 m\_exhaustDamage; // Exhaust damage (percentage)

int8 m\_vehicleFiaFlags; // -1 = invalid/unknown, 0 = none, 1 = green

// 2 = blue, 3 = yellow, 4 = red

float m\_ersStoreEnergy; // ERS energy store in Joules

uint8 m\_ersDeployMode; // ERS deployment mode, 0 = none, 1 = low, 2 = medium

// 3 = high, 4 = overtake, 5 = hotlap

float m\_ersHarvestedThisLapMGUK; // ERS energy harvested this lap by MGU-K

float m\_ersHarvestedThisLapMGUH; // ERS energy harvested this lap by MGU-H

float m\_ersDeployedThisLap; // ERS energy deployed this lap

};

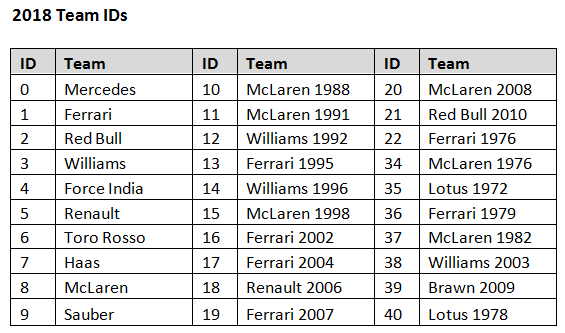
struct PacketCarStatusData

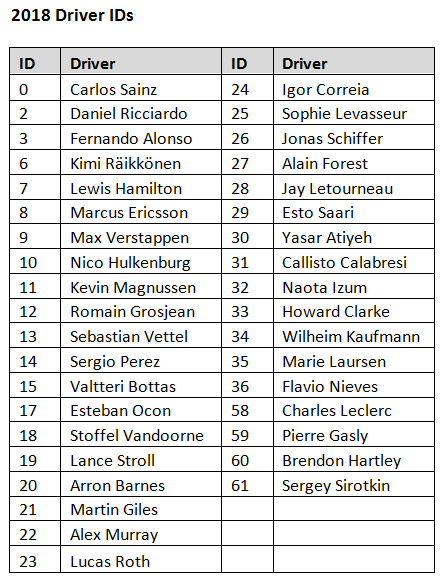
{

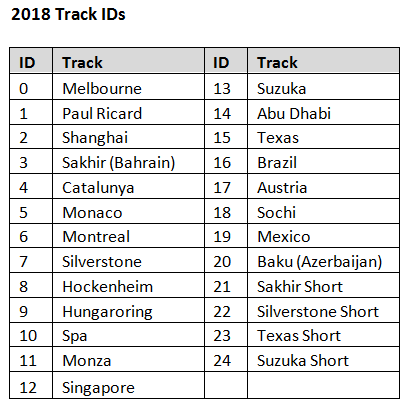
PacketHeader m\_header; // Header

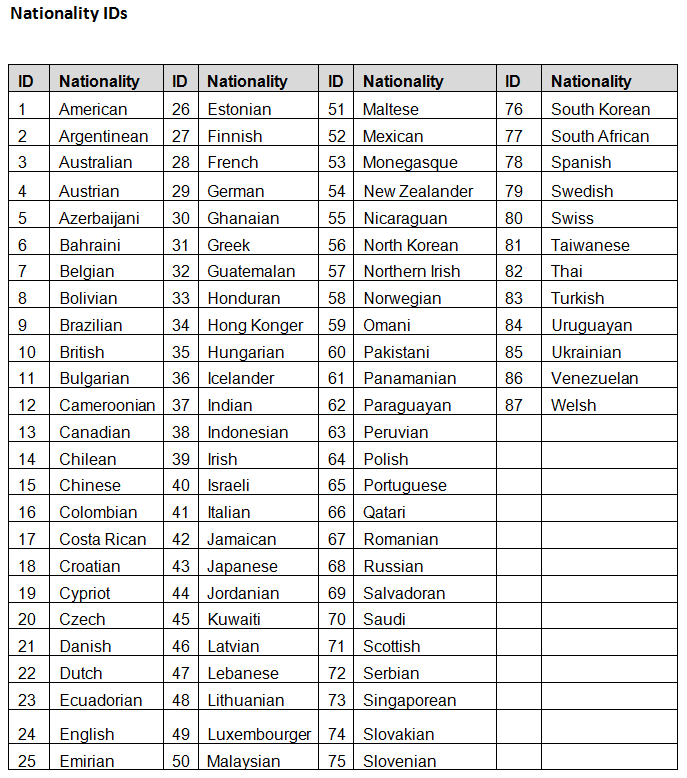
CarStatusData m\_carStatusData[20];

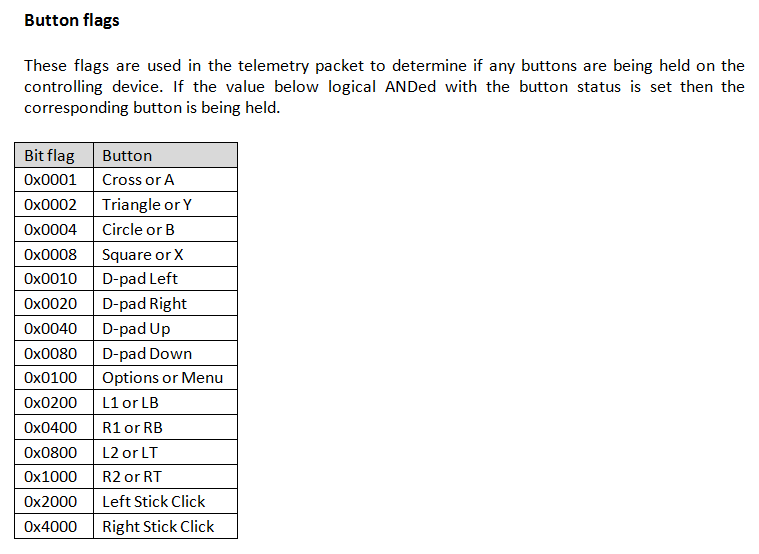
};

[](https://us.v-cdn.net/5021484/uploads/editor/48/y1yaxadoggmk.png)





[](https://us.v-cdn.net/5021484/uploads/editor/0o/9wqezks7xzky.png)

[](https://us.v-cdn.net/5021484/uploads/editor/9b/66rgj8cv225n.png)

**FAQS**

**How do I enable the UDP Telemetry Output?**

In F1 2018, UDP telemetry output is controlled via the menus. To enable this, enter the options menu from the main menu (triangle / Y), then enter the settings menu - the UDP option will be at the bottom of the list. From there you will be able to enable / disable the UDP output, configure the IP address and port for the receiving application, toggle broadcast mode and set the send rate. Broadcast mode transmits the data across the network subnet to allow multiple devices on the same subnet to be able to receive this information. When using broadcast mode it is not necessary to set a target IP address, just a target port for applications to listen on.

**Can I configure the UDP output using an XML File?**

PC users can edit the game’s configuration XML file to configure UDP output. The file is located here (after an initial boot of the game):

   ...\Documents\My Games\<game\_folder>\hardwaresettings\hardware\_settings\_config.xml

You should see the tag:

   <motion>

     ...

     <udp enabled="false" broadcast=”false” ip="127.0.0.1" port="20777" sendRate=”20” format=”2018” />

     ...

   </motion>

Here you can set the values manually. Note that any changes made within the game when it is running will overwrite any changes made manually.

**What is the order of the wheel arrays?**

All wheel arrays are in the following order:

   0 – Rear Left (RL)

   1 – Rear Right (RR)

   2 – Front Left (FL)

   3 – Front Right (FR)

**Do the vehicle indices change?**

During a session, each car is assigned a vehicle index. This will not change throughout the session and all the arrays that are sent use this vehicle index to dereference the correct piece of data.

**What encoding format is used?**

All values are encoded using Little Endian format.

**Is the data packed?**

Yes, all data is packed.

**Will my F1 2017 app still work with F1 2018?**

F1 2018 uses a new format for the UDP data. However, the F1 2017 implementation is still supported by the game and is referred to as the “legacy” format. This should allow most apps implemented using the previous data format to work with little or no change from the developer. To use the old format, please enter the UDP options menu and set “UDP Format” to “legacy”. Specifications for the legacy format can be seen here: <http://forums.codemasters.com/discussion/53139/f1-2017-d-box-and-udp-output-specification/p1>.

**How do I enable D-BOX output?**

D-BOX output is currently supported on the PC platform. In F1 2018, the D-BOX activation can be controlled via the menus. Navigate to **Game Options->Settings->UDP Telemetry Settings->D-BOX** to activate this on your system.

*Advanced PC Users:*It is possible to control D-BOX by editing the games’ configuration XML file. The file is located here (after an initial boot of the game):

...\Documents\My Games\<game\_folder>\hardwaresettings\hardware\_settings\_config.xml

You should see the tag:

  <motion>

    <dbox enabled="false" />

    ...

  </motion>

Set the “enabled” value to “true” to allow the game to output to your D-BOX motion platform. Note that any changes made within the game when it is running will overwrite any changes made manually.

**How can I disable in-game support for LED device?**

The F1 game has native support for some of the basic features supported by some external LED devices, such as the *Leo Bodnar SLI Pro* and the *Fanatec* steering wheels. To avoid conflicts between Codemasters’ implementation and any third-party device managers on the PC platform it may be necessary to disable the native support. This is done using the following **led\_display** flags in the **hardware\_settings\_config.xml**. The file is located here (after an initial boot of the game):

  ...\Documents\My Games\<game\_folder>\hardwaresettings\hardware\_settings\_config.xml

The flags to enabled/disable LED output are:

  <led\_display fanatecNativeSupport="true" sliProNativeSupport="true" />

The **sliProNativeSupport** flag controls the output to SLI Pro devices. The **fanatecNativeSupport** flag controls the output to Fanatec (and some related) steering wheel LEDs. Set the values for any of these to “false” to disable them and avoid conflicts with your own device manager.

Please note there is an additional flag to manually control the LED brightness on the SLI Pro:

  <led\_display sliProForceBrightness="127" />

This option (using value in the range 0-255) will be ignored when setting the **sliProNativeSupport** flag to “false”.

Also note it is now possible to edit these values on the fly via the **Game Options->Settings->UDP Telemetry Settings**menu.