NVIDIA Graphics Card? Crashing? Click here for solution - UPDATED 27/04

Want to report an issue? Have a bug to report? Please look here first!!!!

F1 2020 BETA - *How To Apply*

F1 2018 UDP Specification

By Hoo, August 23, 2018 in Technical Assistance

PREV Page 1 of 12 NEXT

Hoo 101
Posted August 23, 2018

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.

Hoo 101
Posted August 23, 2018

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
};
```

Packet Name	ID	Description	Frequency	Size
Motion	0	Contains motion data for all cars	Menu setting	1341 bytes
Session	1	General data about the session	2 per second	147 bytes
Lap Data	2	Lap time info for all cars in the session	Menu setting	841 bytes
Event	3	Session start or session end	On event	25 bytes
Participants	4	List of participants in the session	Every 5 seconds	1082 bytes
Car Setups	5	Car setup info for cars in the race	2 per second	841 bytes
Car Telemetry	6	Telemetry data for all cars	Menu setting	1085 bytes
Car Status	7	General car status info for all cars	2 per second	1061 bytes

Hoo 101
Posted August 23, 2018

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
    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
};
```

SESSION PACKET

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

Frequency: 2 per second

Size: 147 bytes

```
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
    uint8         m_networkGame;              // 0 = offline, 1 = online
};
```

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

```
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
  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

```
struct PacketEventData
{
  PacketHeader    m_header;             // Header

  uint8           m_eventStringCode[4];   // Event string code, see above
};
```

Event	Code	Description
Session Started	"SSTA"	Sent when the session starts
Session Ended	"SEND"	Sent when the session ends

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

```
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
    uint8      m_nationality;           // Nationality of the driver
    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];
};
```

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

```
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

```
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
};
```

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

```
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];
};
```

Hoo 101
Posted August 23, 2018

Appendices for the various IDs used in the UDP output:

2018 Team IDs

ID	Team	ID	Team	ID	Team
0	Mercedes	10	McLaren 1988	20	McLaren 2008
1	Ferrari	11	McLaren 1991	21	Red Bull 2010
2	Red Bull	12	Williams 1992	22	Ferrari 1976
3	Williams	13	Ferrari 1995	34	McLaren 1976
4	Force India	14	Williams 1996	35	Lotus 1972
5	Renault	15	McLaren 1998	36	Ferrari 1979
6	Toro Rosso	16	Ferrari 2002	37	McLaren 1982
7	Haas	17	Ferrari 2004	38	Williams 2003
8	McLaren	18	Renault 2006	39	Brawn 2009
9	Sauber	19	Ferrari 2007	40	Lotus 1978

(Hoo: IDs have been updated on 10th Sept 2018 as several of them were missing)

2018 Driver IDs

ID	Driver	ID	Driver
0	Carlos Sainz	28	Jay Letourneau
2	Daniel Ricciardo	29	Esto Saari
3	Fernando Alonso	30	Yasar Atiyeh
6	Kimi Räikkönen	31	Callisto Calabresi
7	Lewis Hamilton	32	Naota Izum
8	Marcus Ericsson	33	Howard Clarke
9	Max Verstappen	34	Wilheim Kaufmann
10	Nico Hulkenburg	35	Marie Laursen
11	Kevin Magnussen	36	Flavio Nieves
12	Romain Grosjean	37	Peter Belousov
13	Sebastian Vettel	38	Klimek Michalski
14	Sergio Perez	39	Santiago Moreno
15	Valtteri Bottas	40	Benjamin Coppens
17	Esteban Ocon	41	Noah Visser
18	Stoffel Vandoorne	42	Gert Waldmuller
19	Lance Stroll	43	Julian Quesada
20	Arron Barnes	44	Daniel Jones
21	Martin Giles	58	Charles Leclerc
22	Alex Murray	59	Pierre Gasly
23	Lucas Roth	60	Brendon Hartley
24	Igor Correia	61	Sergey Sirotkin
25	Sophie Levasseur	69	Ruben Meijer
26	Jonas Schiffer	70	Rashid Nair
27	Alain Forest	71	Jack Tremblay

2018 Track IDs

ID	Track	ID	Track
0	Melbourne	13	Suzuka
1	Paul Ricard	14	Abu Dhabi
2	Shanghai	15	Texas
3	Sakhir (Bahrain)	16	Brazil
4	Catalunya	17	Austria
5	Monaco	18	Sochi
6	Montreal	19	Mexico
7	Silverstone	20	Baku (Azerbaijan)
8	Hockenheim	21	Sakhir Short
9	Hungaroring	22	Silverstone Short
10	Spa	23	Texas Short
11	Monza	24	Suzuka Short
12	Singapore		

Nationality IDs

ID	Nationality	ID	Nationality	ID	Nationality	ID	Nationality
1	American	26	Estonian	51	Maltese	76	South Korean
2	Argentinean	27	Finnish	52	Mexican	77	South African
3	Australian	28	French	53	Monegasque	78	Spanish
4	Austrian	29	German	54	New Zealander	79	Swedish
5	Azerbaijani	30	Ghanaian	55	Nicaraguan	80	Swiss
6	Bahraini	31	Greek	56	North Korean	81	Taiwanese
7	Belgian	32	Guatemalan	57	Northern Irish	82	Thai
8	Bolivian	33	Honduran	58	Norwegian	83	Turkish
9	Brazilian	34	Hong Konger	59	Omani	84	Uruguayan
10	British	35	Hungarian	60	Pakistani	85	Ukrainian
11	Bulgarian	36	Icelander	61	Panamanian	86	Venezuelan
12	Cameroonian	37	Indian	62	Paraguayan	87	Welsh
13	Canadian	38	Indonesian	63	Peruvian		
14	Chilean	39	Irish	64	Polish		
15	Chinese	40	Israeli	65	Portuguese		
16	Colombian	41	Italian	66	Qatari		
17	Costa Rican	42	Jamaican	67	Romanian		
18	Croatian	43	Japanese	68	Russian		
19	Cypriot	44	Jordanian	69	Salvadoran		
20	Czech	45	Kuwaiti	70	Saudi		
21	Danish	46	Latvian	71	Scottish		
22	Dutch	47	Lebanese	72	Serbian		
23	Ecuadorian	48	Lithuanian	73	Singaporean		
24	English	49	Luxembourger	74	Slovakian		
25	Emirian	50	Malaysian	75	Slovenian		

Button flags

These flags are used in the telemetry packet to determine if any buttons are being held on the controlling device. If the value below logical ANDed with the button status is set then the corresponding button is being held.

Bit flag	Button
0x0001	Cross or A
0x0002	Triangle or Y
0x0004	Circle or B
0x0008	Square or X
0x0010	D-pad Left
0x0020	D-pad Right
0x0040	D-pad Up
0x0080	D-pad Down
0x0100	Options or Menu
0x0200	L1 or LB
0x0400	R1 or RB
0x0800	L2 or LT
0x1000	R2 or RT
0x2000	Left Stick Click
0x4000	Right Stick Click

Hoo 101
Posted August 23, 2018

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.