Overview

UDP telemetry is available for export for certain game data.

Getting Started

1. Start EA SPORTS™ WRC, allowing it to progress to the first interactive screen (i.e. click Start).
2. Quit the title and navigate to the newly generated folder “Documents/My Games/WRC/telemetry/”
3. Edit the config.json file to enable telemetry as you see fit (for reference see below). See the readme folder and below for documentation.
4. Re-launch EA SPORTS™ WRC, which will reload the config.json file (and check log.txt for errors).

Channel Data

Telemetry is available as channel data. For reference, this title’s complete channel data can be found in readme/channels.json: this file is automatically regenerated when the title starts.

readme/channels.json – READ ONLY:

- versions:schema – The json file schema version, which changes when the json structure has been modified. Modifications may or may not be compatible between versions, so any tools/scripts reading this file should check this version.
- versions:data – Incremented when channels are added/removed/modified.
- channels – The list of available telemetry channels.
- channels:id – A unique identifier associated with the telemetry channel. This channel ID is referenced in packet structure files, for adding telemetry data to packets.
- channels:type – The data type associated with the given telemetry data.
  All data values are encoded using Little Endian format and packed in order of ID supplied in the packet structure (See **Packet Structure** for more information). See **Channel Types** for the list of available types.
- channels:units – A unit descriptor to help describe the value stored in the channel. Can be used by tool chains to display alongside the value, e.g. for debugging purposes.
- channels:description - A descriptor for what’s stored in the telemetry channel, useful for understanding how the data can be displayed/interpreted at its end-points.

Scopes & Packets

A packet comprises channel IDs in the order that telemetry data is packetted.

The title sends packets at specific times over the lifetime of a packet “scope”:

- <scope_id>_start – Single packet sent at the start of the scope.
- <scope_id>_update – Multiple packets, each sent whenever the scope is updated.
- <scope_id>_end – Single packet sent at the end of the scope.
- <scope_id>_pause – Single packet sent when the scope is paused.
- <scope_id>_resume – Single packet sent when the scope is unpaused.

Currently, the title supports a single scope called “Session” which covers driving through a rally stage. More scopes may be added in future releases/patches. The current list of supported scopes are found in readme/packets.json: this file is automatically regenerated when the title starts.

readme/packets.json – READ ONLY:

- versions:schema – The json file schema version, which changes when the json structure has been modified. Modifications may or may not be compatible between versions, so any tools/scripts reading this file should check this version.
- versions:data – Incremented when packets/scopes are added/removed/modified.
- packets – The list of available packets.
- packets:id – A unique identifier associated with the packet. This packet ID is referenced in packet structure files and the telemetry configuration file.
- packets:fourCC – A short fourCC unique identifier for the current packet, sent in the telemetry channel “packet_uid”.

Packet Structure

Each packet structure is defined in the files “udp/<structure_id>.json”.

Each file defines the telemetry channels (and their order) contained within a given packet.

For reference, the title supplies three packet structures:

- udp/custom1.json – User editable and loaded by the title.
- readme/udp/wrc.json – A predefined read-only file, exported when the application starts, showing the default packet layout.
- readme/udp/wrc_experimental.json - A predefined read-only file, exported when the application starts, showing the default experimental packet layout.

Any number of custom packet structures can be dropped into the udp/ folder, as supplied by third parties or custom created by end users.

udp/custom1.json

- versions:schema – The json file schema version, which changes when the json structure has been modified. Modifications may or may not be compatible between versions, so any tools/scripts reading this file should be checking this version.
- versions:data – Incremented when the data is modified.
- id – A unique identifier associated with the packet structure. (We recommend using the file name excluding its extension.)
- header:channels – The list of common channel IDs to include with (and prefix) every packet. E.g. If “packet_uid” is defined here, each packet (“session_update”, “session_start”, etc) will have “packet_uid” prefixed before its dedicated channels found under “packets:channels”.
- packets – The list of available packet structures.
- packets:id – A unique identifier associated with a packet. See **Packets** for more information.
- packets:channels – The list of channel IDs in order of serialisation: see the channels.json file. This list can be defined/read at runtime, and is fully customisable by the end user.

Config

For each packet, file config.json assigns a packet structure, UDP end-points, and other miscellaneous settings. (This replaces the file hardware_config.xml found in previous titles.)

Each end-point can be targeted with a specific IP/port, and enabled or delayed independently.

This file is generated once when the title is first loaded and then parsed on subsequent runs, unless patched (a new config will be generated aside a backup config).

config.json

- versions:schema – The json file schema version, which changes when the json structure has been modified. Modifications may or may not be compatible between versions, so any tools/scripts reading this file should check this version.
- udp:packets – The list of packet assignments.
- udp:packets:structure – The packet structure ID assigned to the packet. This ID can be of a built-in file (see readme/udp/), or of a custom user file (see udp/).
- udp:packets:packet – The packet ID. See **Packets** for more information.
- udp:packets:ip – The ip4 or ip6 target end-point receiving the UDP packet.
- udp:packets:port – The target end-point port receiving the packet. Note: To avoid socket buffer limits causing dropped/lost packets, we recommend assigning <scope_id>_start, <scope_id>_end, <scope_id>_pause, and <scope_id>_resume (which receive data infrequently) a different port than <scope_id>_update (which receives data frequently).
- udp:packets:frequencyHz – -1 (no limit) / 0 (Disabled, no packets sent) / 1+ (Frequency Limit in Hz) - Limits the number of packets sent within the specified Frequency Limit. I.e. If the game is running at 240fps (Hz), with a limit of 60Hz, packets will be received at an average rate of 1 packet per 16ms at 60Hz. The slower you receive the data, the more reliable the UDP packets will be but at the cost of being less up-to-date.
- udp:packets:enabled – Enable/disable packet transmission.
- lcd:bDisplayGears – For supported LED displays, display current gear index (true) or vehicle speed (false).
- dBox:bEnabled – Enable/disable D-Box telemetry output.
- dBox:bLegacyMode – Deprecated at release.

Log.txt

Comprises basic status and error logging. After making any changes to packets and structures, telemetry channels, UDP ports, or facing in-game issues with telemetry we recommend checking this file for errors.

Channel Types

- boolean – single bit
- uint8 – Unsigned 8-bit integer
- int8 – Signed 8-bit integer
- uint16 – Unsigned 16-bit integer
- int16 – Signed 16-bit integer
- uint32 – Unsigned 32-bit integer
- int32 – Signed 32-bit integer
- uint64 – Unsigned 64-bit integer
- int64 – Signed 64-bit integer
- float32 – 32-bit floating point
- float64 – 64-bit floating point
- fourcc – 4 character string identifier (not null terminated)

FAQs

- Can I edit any of the files in the readme/ folder?
    - No. These files are generated every time the application is started, and are read-only.
- Can I edit the wrc or wrc_experimental configs?
    - Yes & No. You cannot edit readme/udp/ files directly (they are not read by the title).
      You can copy these files to udp/ and rename them to <your custom name>.json and edit the packet structure “id” to match. After that, you can edit any of the channels.
- I don’t have a custom1.json file in my udp/ folder. How do I fix this?
    - Delete the udp/ folder and this folder will be re-generated when you next launch the title.
- My config.json file is corrupt and will not load: what do I do?
    - Rename the file to config.backup.json and launch the title. A new config.json will be generated that you can re-edit using config.backup.json as reference.
- How often are update packets sent?
    - These are sent every frame (if -1 is set) unless a frequency lower than your current framerate is specified.
- Will wrc.json work with future patches?
    - To retain backwards compatibility in future patches, we will add new packet structures on adding new data (e.g. wrc_<version>.json). Older structures will continue to work, unless stated otherwise.
- What is the difference between wrc and wrc_experimental packet structures?
    - wrc_experimental is the same data as the wrc packet structure in a format that reduces packet sizes by only sending data that changes. E.g. Max Engine RPM will not change during an update so is only sent once. This is experimental as it could be prone to packet loss on slow network connections due to the lossy nature of UDP packets. Please provide feedback on this packet structure layout via the community forums.

Legal Notice