

Session crossings - JSON download

The MS Timing team portal lets a race team download, for any session of its category at the event, every crossing of the circuit's on-track timing lines from the session's warm-up to the end of the session, as one JSON file. This document describes that file exactly, so that an engineer, a script or an AI model can read it and turn it into lap times, sector times, microsector times, gaps and running order.

The file is plain UTF-8 JSON (one object). Times are ISO 8601 with an explicit offset. Numbers are JSON numbers. Fields with no value are left out rather than written as null.

1. The top-level object

event	string	The event (meeting) name, e.g. "BATHURST 1000".
session	string	The session name, e.g. "Race 30 - Qualifying 1".
category	string	The category, e.g. "Supercars".
sessionStartUtc	date-time	When the session was started (warm-up). Crossings start here.
sessionEndUtc	date-time	When the session was stopped. Absent while the session is still running.
generatedUtc	date-time	When this file was produced.
note	string	A one-line human summary of the file.
lines	array	The timing lines of the circuit that appear in this file (section 2).
cars	array	The cars in the session (section 3).
crossings	array	Every crossing, in time order (section 4).

2. lines[] - the timing lines

The on-track timing lines, in track order starting at the lap line. Each is a loop in the road that records a car passing over it.

loop	integer	The line's loop number - the key used by crossings[].loop.
name	string	Its name, e.g. "Main Line", "Pos 1", "Int 1".
distanceMetres	number	Distance from the lap line along the racing line, in metres. The lap line is 0.
lapLine	boolean	true for the line that completes a lap (the start / finish line).

Consecutive lines bound a MICROSECTOR: the time between a car crossing line N and line N+1 is that microsector's time. The official sectors are the lines named "Int" (intermediates) plus the lap line. Pit-lane loops and track-limits loops are deliberately not included.

3. cars[]

car	integer	The race number - the key used by crossings[].car.
driver	string	The nominated driver ("First Last"; a surname can contain spaces, e.g. "Jack Le Brocq").
team	string	The team / entrant name, when set.

4. crossings[] - the data

One object per crossing, ordered by time (oldest first). A car crossing a line produces one crossing.

car	integer	Race number of the car.
loop	integer	Loop number of the line crossed (see lines[]).
line	string	The line's name, for convenience.
utc	date-time	The exact crossing time from the timing decoder, UTC, to 1/10,000 s.
local	string	The same instant as the venue's time of day, HH:mm:ss.fff.
lap	integer	Present ONLY on a lap-line crossing that the timing system counted as

disallowed	boolean	completing a lap: the number of the lap just completed (1 = first lap). Present (true) when race control disallowed the crossing - ignore it for times.
beforeStart	boolean	Present (true) when the crossing happened before the start (grid / formation lap) - not part of a timed lap.

5. Turning crossings into times

Lap time: for one car, the time between two consecutive lap-line crossings that both carry a lap number: $\text{lapTime}(n) = \text{utc}(\text{lap } n) - \text{utc}(\text{lap } n-1)$. Lap 1 in a race is timed from the green flag, which is not in this file - use the crossing with lap = 1 against the start time you know, or start at lap 2.

Microsector time: for one car, the time between consecutive crossings of ADJACENT lines (line N then line N+1 in lines[] order, wrapping from the last line back to the lap line). If a crossing is missing (a loop did not read the car), the gap spans two microsectors - skip it rather than split it.

Sector time: the same between the official sector lines (the "Int" lines and the lap line).

Ignore crossings with disallowed = true or beforeStart = true. A car that goes through the pit lane leaves the circuit's lines between pit entry and pit exit; the lap that includes the stop is an in-lap / out-lap and is not representative.

Running order at any instant: sort cars by laps completed (highest first), then by the time of their latest lap-line crossing (earliest first).

6. Example

```
{
  "event": "BATHURST 1000",
  "session": "Race 30 - Qualifying 1",
  "category": "Supercars",
  "sessionStartUtc": "2026-10-10T04:05:00+00:00",
  "sessionEndUtc": "2026-10-10T04:35:12+00:00",
  "lines": [
    { "loop": 60, "name": "Main Line", "distanceMetres": 0, "lapLine": true },
    { "loop": 1, "name": "Pos 1", "distanceMetres": 545, "lapLine": false },
    { "loop": 20, "name": "Int 1", "distanceMetres": 2414, "lapLine": false }
  ],
  "cars": [ { "car": 17, "driver": "Brodie Kostecki", "team": "Dick Johnson Racing" } ],
  "crossings": [
    { "car": 17, "loop": 60, "line": "Main Line", "utc": "2026-10-10T04:12:01.2345+00:00", "local": "15:12:01.234", "lap": 3 },
    { "car": 17, "loop": 1, "line": "Pos 1", "utc": "2026-10-10T04:12:13.9875+00:00", "local": "15:12:13.987" }
  ]
}
```

Here car 17 completed lap 3 at 04:12:01.2345 UTC and took 12.753 s from the lap line to Pos 1 (microsector 1).

7. For an AI model reading this file

Treat crossings[] as an append-only event log keyed by (car, loop, utc). Join it to lines[] on loop to get track position and to cars[] on car to get the driver. Compute times only from differences of utc values of the SAME car. Never infer pit-lane or track-limits information - it is not in this file by design.