# 1) Información: cómo leer/controlar TSC desde fuera y dentro del juego

## 1.1. Vías de acceso a datos/controles

- **RailDriver API (raildriver.dll / raildriver64.dll)**: interfaz oficial para hardware RailDriver que también permite **leer** (velocidad, posición aproximada, estados) y **controlar** (acelerador, freno, reverser...) desde apps externas.
- Wrappers y ejemplos:
  - Python: `py-raildriver` (GitHub / PyPI)
  - C/Python: `alios/raildriver` (ejemplo de bindings)
  - **REST server** sobre RailDriver: `YoRyan/railsim-remote` (expone endpoints HTTP)
- Docs/soporte: RailDriver (PI Engineering) – descargas y manuales.

- **Scripting LUA interno (escenarios / locomotoras / señales)**: permite llamar funciones del motor como `PlayerEngine:GetSpeed()` y `PlayerEngine:GetNextSpeedLimit(dir, offset)` para consultar HUD/track data desde **scripts dentro del juego**. Útil para prototipos, overlays internos y pruebas de lógica (p. ej., frenar por próximo límite).

- Referencias comunitarias (no oficiales): ChrisTrains Dev Docs; foros TrainSimDev; hilos en Steam/DTG.

- **LogMate**: logger oficial de TSC (diagnóstico). Útil para **parsear eventos** (errores de escenarios, señalización, carga de assets) y construir telemetrías mínimas si se habilitan mensajes relevantes.

- **SERZ / Blueprints**: `serz.exe` convierte `.bin ⇄ .xml` para leer **propiedades físicas** y de control de material rodante y señales. **TS-Tools** (antes RW Tools) abre `.bin` directamente y ayuda a localizar masas, frenos, longitudes, límites, etc.

- **Interfaz RailDriver & Joystick (terceros)**: apps que mapean joysticks y crean overlays (velocidad, límite, indicadores) sirven como referencia de **qué datos expone el sim** y cómo integrarlos con hardware casero.

  Nota: TSC **no expone** una API de red pública tipo socket; el camino más estable hoy es RailDriver + scripting/LUA in-game + utilidades de análisis (LogMate, SERZ/TS-Tools).

## 1.2. Enlaces útiles (curados)

- RailDriver (descargas/soporte):
  https://raildriver.com/support/downloads.php
  Manual RD: https://raildriver.com/support/manuals.php
- Wrapper Python:
  https://github.com/piotrkilczuk/py-raildriver
  https://pypi.org/project/py-raildriver/

- REST sobre RailDriver:
  https://github.com/YoRyan/railsim-remote
- Ejemplo bindings Python:
  https://github.com/alios/raildriver
- Docs no oficiales LUA/Dev:
  https://www.christrains.com/tscdevdocs/home.html
  (Señalización: https://www.christrains.com/tscdevdocs/reference-manual/signalling-guide/train-simulator-signalling.html)
- Foros/threads clave (LUA, funciones, límites):
  Trainsimdev – GetNextSpeedLimit: https://www.trainsimdev.com/forum/viewtopic.php?f=30&t=451
  DTG/Steam hilos sobre LUA y docs: (varios, ver research/refs)
- LogMate (cómo generar informe):
  https://dovetailgames.freshdesk.com/support/solutions/articles/80000895937-how-do-i-create-a-logmate-report-
- SERZ y TS-Tools:
  SERZ: en carpeta `RailWorks` (usa también `SerzMaster.exe`)
  TS-Tools: versiones esp mirror comunitario (SimTogether, Railworks America, ATS)

## 1.3. Snippets de arranque

**Python + RailDriver (leer velocidad km/h):**

```
from raildriver import RailDriver
rd = RailDriver()
rd.setRailSimConnected(True)
rd.setRailDriverConnected(True)
# GetSpeed suele devolver m/s
v_ms = rd.getCurrentSpeed()
kmh = v_ms * 3.6
print(f"Velocidad: {kmh:.2f} km/h")
```

**LUA (HUD/escenario): siguiente límite y distancia de consulta:**

```
-- dir: 0 hacia delante, 1 hacia atrás; offset: metros desde el tren
local next_limit = SysCall("PlayerEngine:GetNextSpeedLimit", 0, 0)
local speed_ms  = SysCall("PlayerEngine:GetSpeed")
-- speed_ms está en m/s
```

**SERZ (VS Code):** - Extensión “RailWorks Serz integration” → atajo `SHIFT+ALT+Q` para bin⇄xml.

---

# 2) Normas y estructura del proyecto

## 2.1. Principios

1. **Separación estricta** de capas: *ingestión* (lectura de datos), *modelo* (física/curvas), *control* (actuadores), *UI/telemetría*, *I/O* (logs/CSV).
2. **Determinismo**: toda decisión de la IA debe quedar trazada en CSV/JSON (inputs, cálculo, output).
3. **Modularidad por tren/ruta**: perfiles y `*.json` por material/servicio (masas, frenos, límites propios, AFB, PZB/LZB si aplica).
4. **Pruebas automáticas** para cada módulo (fixtures con telemetría sintética y real).
5. **Documentación diaria** breve y obligatoria.

## 2.2. Estructura de carpeta (propuesta)

```
/tsc-ai
  /ingestion
    raildriver_client.py        # wrapper estable
    logmate_parser.py           # parser filtrado
    serz_scan.py                # extrae físicas de blueprints
  /physics
    braking_curves.py           # integra Excel ERA v5.1 (CSV export)
    adhesion_models.py          # μ(v, clima), gradiente, etc.
  /control
    autopilot.py                # lazo PID/MPPI; target v, control freno/
acelerador
    speed_planner.py            # plan por próximo límite + estaciones
  /profiles
    BR146.json                  # masas, frenos, límites, notch map
    *-route-overrides.json      # ajustes por ruta
  /ui
    overlay.py                  # HUD debug
    dashboard_streamlit.py      # telemetría en vivo
  /data
    laps.csv / runs/*.csv       # logs por run
    blueprints/*.xml            # dump SERZ
  /tests
    test_braking_curves.py
    test_controller.py
  /docs
    diario.md                   # ver plantilla (Sección 3)
    roadmap.md                  # ver Sección 4
  /research
    refs.md                     # enlaces y notas
```

## 2.3. Convenciones de código

- Python ≥ 3.11, estilo `ruff + black`; tipado `mypy` estricto.

- Nombres de señales/controles en inglés; comentarios bilingües corto.
- Unidades **SI** internas (m, s, m/s); conversión en I/O.
- CSV con `;` separador, `.` decimal; timestamps ISO8601 local Europe/Madrid.

### 2.4. Telemetría mínima (MVP)

- `t`: tiempo, `v_ms`, `limit_next_kmh`, `dist_next_limit_m`, `gradiente`, `throttle`, `brake_notch`, `service_pressure`, flags (SIFA/PZB si disponibles).

---

# 3) Documentación diaria (plantilla)

Copia y pega por día en `docs/diario.md`:

```
## 2025-09-04 (Jue) – Sprint P0 Arranque
- Objetivo del día: ☐ instalar RailDriver ☐ leer v_ms ☐ export laps.csv
- Hechos clave:
  - [ ] py-raildriver funcionando (lectura velocidad)
  - [ ] Estructura repo creada
- Decisiones:
  - Vector de estado MVP: {v_ms, limit_next, dist_next}
- Problemas/Bloqueos:
  - …
- Métricas/Runs:
  - run_2025-09-04_1.csv (duración 12:31, max 120.4 km/h)
- Mañana:
  - Parser LogMate y primer overlay (HUD velocidad/limit)
```

---

# 4) Hoja de ruta (v1, editable)

**P0 – Boot (2–3 días)** - Instalar RailDriver + wrapper Python. - Script lectura continua → `runs/*.csv` + dashboard básico (velocidad y límite siguiente manual).

**P1 – Telemetría HUD/LUA (1–2 semanas)** - PoC LUA in-game para `GetNextSpeedLimit` y `GetSpeed` (si viable por escenario). - Fallback: estimador de límite por señales/blueprints si no hay acceso directo.

**P2 – Modelo de frenado (1–2 semanas)** - Integrar `braking_curves.py` con **ERA v5.1** (export CSV desde tu .xlsm).
- Validar `base/deg` por consist y gradiente.

**P3 – Controlador** - Lazo **feed-forward** + **PID** (target en función de distancia a próximo límite y tiempos de parada).
- Añadir márgenes de confort y anticipo (suavizado + seguridad).

**P4 – Señalización y estaciones** - Lectura de estados (si es posible) y planificador de paradas.

**P5 – Endurecer & empaquetar** - Perfiles por tren/ruta, tolerancias, grabación completa, README + vídeos.

---

## Checklist inicial (tick-box)

- [ ] RailDriver instalado y calibrado
- [ ] `py-raildriver` lee velocidad real
- [ ] `runs/*.csv` se genera
- [ ] Plantilla `docs/diario.md` creada y en uso
- [ ] `research/refs.md` con enlaces y notas
- [ ] Roadmap `docs/roadmap.md` actualizado