Guía de Integración del Protocolo Rinho

> **Versión:** 1.2
> **Fecha:** 2026-05-06
> **Audiencia objetivo:** Equipo de ingeniería de Flespi / IA de integración de protocolos
> **Familia de protocolo:** TAIP (Trimble ASCII Interface Protocol) con extensiones Rinho
> **Transporte:** UDP y TCP

Tabla de contenidos

1. Descripción general

Los dispositivos Rinho usan un protocolo basado en **TAIP (Trimble ASCII Interface Protocol)** con extensiones propietarias. El protocolo soporta:

- **Mensajes ASCII** -- Reportes de posición legibles por humanos, consultas de versión, keep-alive, comandos
- **Comunicación bidireccional** -- El servidor puede enviar comandos de configuración y recibir confirmaciones
- **Numeración de mensajes** -- Entrega confiable con mecanismo de ACK/retransmisión
- **Telemetría extendida** -- Sensores de temperatura, batería de respaldo, horómetro, iButton, bus CAN (OBD-II/J1939)

Flujo de comunicación del dispositivo

1. El dispositivo envía un **reporte de posición** (por ej. RCQ) con un número de mensaje (`#MSGNUM`).
2. El servidor recibe, persiste y responde con un **ACK** que repite el mismo `#MSGNUM`.
3. Si no se recibe ACK en 7 segundos, el dispositivo **retransmite** (hasta 4 reintentos).
4. El servidor puede enviar un **comando** al dispositivo con su propio `#MSGNUM`.
5. El dispositivo ejecuta el comando y responde con una **confirmación** donde `msgNum = original | 0x8000`.
6. El dispositivo envía mensajes periódicos de **keep-alive** (KA); estos no requieren ACK.

---

2. Conexión y transporte

Propiedad	Valor
Transporte	**UDP** y **TCP**
Puertos por defecto	Configurables (típicamente 4031 primario, 4034 secundario)
Codificación	ASCII (7-bit)
Tamaño máximo de paquete	~4096 bytes (límite práctico)
Keep-alive	El dispositivo envía `>KA;ID=XXXX;*CS<` cada 60 segundos (configurable)

Modo UDP

- Sin conexión -- el dispositivo envía desde un puerto dinámico.
- El servidor debe mantener una **tabla NAT** (mapeo deviceId a IP:puerto) tomada de la dirección de origen del último paquete recibido.
- Los comandos se envían de vuelta al último IP:puerto conocido del dispositivo a través del **mismo socket UDP** que recibió el último mensaje del dispositivo.

Modo TCP

- El dispositivo abre una conexión TCP persistente al servidor.
- **El framing es idéntico** al de UDP: los mensajes se delimitan con `>` (inicio) y `<` (fin).
- Como TCP es un protocolo de stream, el parser debe **buscar los límites** `>...<` en el flujo de bytes. Pueden llegar varios mensajes en un mismo segmento TCP, o un mensaje puede abarcar varios segmentos. Hay que ir buffereando los bytes entrantes y extraer frames completos.
- Los comandos se envían de vuelta por la **misma conexión TCP** que estableció el dispositivo.
- No hace falta tabla NAT en TCP porque la conexión es persistente.
- El dispositivo intentará reconectarse si se cae la conexión.

Tabla NAT (solo UDP)

Como UDP no tiene conexión, el servidor debe trackear el último IP:puerto conocido de cada dispositivo para poder enviarle comandos. Hay que actualizar este mapeo cada vez que se recibe un paquete del dispositivo. Conviene expirar entradas que no se hayan visto por mucho tiempo (por ej. 24 horas).

---

3. Estructura de mensajes (framing)

Todos los mensajes TAIP siguen esta estructura:

```
>BODY[;#MSGNUM];ID=DEVICEID;*CHECKSUM<
```

Componente	Requerido	Formato	Descripción
`>`	Sí	Char	Delimitador de inicio
`BODY`	Sí	ASCII	Contenido del mensaje (varía según el tipo)
`;#MSGNUM`	Opcional	`;#` + 4 dígitos hex	Número de mensaje (mayúsculas, ej: `#0042`)
`;ID=DEVICEID`	Sí	`;ID=` + alfanumérico/guiones	Identificador del dispositivo (típicamente el IMEI)
`;*CHECKSUM`	Sí	`;*` + 2 dígitos hex	Checksum XOR (mayúsculas)
`<`	Sí	Char	Delimitador de fin

Múltiples mensajes por paquete: Un mismo datagrama UDP o segmento TCP puede contener varios mensajes TAIP concatenados. Siempre escanear TODOS los frames `>...<` en los datos recibidos.

Ejemplo:

```
>RCQ28060426153025-3462000-05838000045180A203128000000110000050A100512;#0001;ID=860012345678901;*5E<
```

Manejo de mensajes mal formados

- Si la validación de checksum falla, **descartar el mensaje en silencio** (no enviar ACK).
- Si el cuerpo del mensaje es demasiado corto o no se pueden parsear los campos, descartar y loguear para debugging.
- Los tipos de mensaje desconocidos (cuyo body no coincide con ningún prefijo conocido) deberían loguearse pero no recibir ACK.

---

4. Cálculo de checksum

XOR de todos los bytes desde `>` hasta `*` (inclusive). Los dos dígitos hex del checksum y el `<` de cierre **se excluyen** del cálculo.

Algoritmo:

1. Tomar el mensaje parcial desde `>` hasta `*` (inclusive): `>ACK;#0001;ID=860012345678901;*`
2. Hacer XOR de cada byte de esa cadena: `'>' ^ 'A' ^ 'C' ^ 'K' ^ ';' ^ '#' ^ '0' ^ ... ^ ';' ^ '*'`
3. Formatear el resultado como 2 dígitos hex en mayúscula: `"5E"`
4. Agregar el checksum y el delimitador de cierre: `>ACK;#0001;ID=860012345678901;*5E<`

Pseudocódigo:

```
def calculate_checksum(partial_message: str) -> str:
    """partial_message incluye desde '>' hasta '*' (inclusive).
    Los dígitos del checksum y el '<' NO se incluyen."""
    checksum = 0
    for byte in partial_message.encode('ascii'):
        checksum ^= byte
    return f"{checksum:02X}"
```

---

5. Reportes de posición ASCII (familia RCQ)

Formato del mensaje

```
>R{TIPO}{66 chars de datos base}[sufijo extendido][;segmentos extra];#MSGNUM;ID=DEVICEID;*CHECKSUM<
```

Los **datos base** después del prefijo de 3 chars (`R` + código de tipo de 2 chars) son siempre **66 caracteres** con campos posicionales fijos.

Campos base (66 caracteres, posicionales)

Offset	Largo	Campo	Formato	Descripción	Conversión
0-1	2	ReportID	Hex `AA`	Secuencia del reporte (00-FF)	Tal cual
2-3	2	Day	Decimal `DD`	Día del mes	Entero
4-5	2	Month	Decimal `MM`	Mes (01-12)	Entero
6-7	2	Year	Decimal `YY`	Año (00-99)	2000 + YY
8-9	2	Hour	Decimal `HH`	Hora UTC (00-23)	Entero
10-11	2	Minute	Decimal `MM`	Minuto (00-59)	Entero
12-13	2	Second	Decimal `SS`	Segundo (00-59)	Entero
14-21	8	Latitude	Signado `+NNNNNNN` o `-NNNNNNN`	Latitud cruda	**valor / 100000** = grados
22-30	9	Longitude	Signado `+NNNNNNNN` o `-NNNNNNNN`	Longitud cruda	**valor / 100000** = grados
31-33	3	Speed	Decimal `NNN`	Velocidad	**km/h** (directo, NO nudos)
34-36	3	Course	Decimal `CCC`	Rumbo	Grados (0-359)
37-38	2	Inputs	Hex `HH`	Bitmask de entradas digitales	Ver "Bitmask de Entradas" abajo
39-40	2	Outputs	Hex `II`	Bitmask de salidas digitales	Ver "Bitmask de Salidas" abajo
41-43	3	Voltage	Decimal `JJJ`	Alimentación principal	**valor / 10** = Volts
44-51	8	Odometer	Hex `KKKKKKKK`	Distancia total	Hex a decimal = **metros**
52	1	GPSPower	`0` o `1`	Energía del módulo GPS	0=Apagado, 1=Encendido
53	1	GPSMode	`2` o `3`	Modo de fix GPS	2=2D, 3=3D
54-55	2	PDOP	Decimal `NN`	Position dilution of precision	Entero (más bajo = mejor, rango 0-99)
56-57	2	Satellites	Decimal `OO`	Cantidad de satélites	Entero
58-61	4	GPSAge	Hex `PPPP`	Tiempo desde el último fix	Hex a decimal = **segundos**
62	1	ModemPower	`0` o `1`	Energía del módem	0=Apagado, 1=Encendido
63	1	GSMStatus	`0` - `5`	Estado de registro	Ver tabla abajo
64-65	2	CSQ	Decimal `SS`	Intensidad de señal	0-30 dBm, 99=sin señal

Bitmask de Entradas (2 chars hex = 8 bits)

Bit	Nombre	Descripción
7	IGN	**Ignición** (1=Encendida, 0=Apagada) -- el bit más importante
6	IN06	Entrada digital 6
5	IN05	Entrada digital 5
4	IN04	Entrada digital 4
3	IN03	Entrada digital 3
2	IN02	Entrada digital 2
1	IN01	Entrada digital 1
0	IN00	Entrada digital 0

Extracción de ignición: `ignition = (inputs_byte & 0x80) != 0`

Bitmask de Salidas (2 chars hex = 8 bits)

Bit	Nombre	Descripción
0	XP00	Salida digital 0
1	XP01	Salida digital 1
2	XP02	Salida digital 2
3-7	--	Reservados (sin uso)

Estado de registro GSM

Valor	Significado
0	No registrado
1	Registrado (red propia)
2	Buscando
3	Denegado
4	Desconocido
5	Roaming

Ejemplo de parseo

```
Crudo: >RCQ28060426153025-3462000-
05838000045180A203128000000110000050A100512;#0001;ID=860012345678901;*5E<

Prefijo:     R + CQ (reporte de posición estándar)
Data[0:2]:   "28"          -> ReportID = 0x28
Data[2:14]:  "060426153025" -> 2026-04-06 15:30:25 UTC
Data[14:22]: "-3462000"    -> Latitud   = -3462000 / 100000 = -34.62000 grados
Data[22:31]: "-05838000"   -> Longitud  = -5838000 / 100000 = -58.38000 grados
Data[31:34]: "045"         -> Velocidad = 45 km/h
Data[34:37]: "180"         -> Rumbo     = 180 grados
Data[37:39]: "A2"          -> Entradas  = 0xA2 = 10100010 -> IGN=ON, IN05=ON, IN01=ON
Data[39:41]: "03"          -> Salidas   = 0x03 = XP00=ON, XP01=ON
Data[41:44]: "128"         -> Voltaje   = 128 / 10 = 12.8 V
Data[44:52]: "00000011"    -> Odómetro  = 0x11 = 17 metros
Data[52:53]: "0"           -> GPS Power = Apagado
Data[53:54]: "3"           -> GPS Mode  = 3D
Data[54:56]: "05"          -> PDOP      = 5
Data[56:58]: "10"          -> Satélites = 10
Data[58:62]: "0512"        -> GPS Age   = 0x0512 = 1298 segundos
Data[62:63]: "1"           -> Módem     = Encendido
Data[63:64]: "1"           -> GSM       = Registrado (red propia)
Data[64:66]: "15"          -> CSQ       = 15 dBm
MsgNum:      "0001"
DeviceID:    "860012345678901"
Checksum:    "5E"
```

---

6. Tipos de reporte extendidos

Todos los tipos extendidos comparten la misma estructura base de 66 chars. Después de la base, cada tipo agrega campos específicos como sufijo.

Códigos de tipo

Código	Prefijo completo	Descripción	Campos extra después de la base
CQ	RCQ	Posición estándar (GPS filtrado)	Ninguno
CP	RCP	Posición (GPS sin filtrar, fix crudo)	Ninguno
CR	RCR	Posición + 1 temperatura	7 chars
CV	RCV	Posición + 2 temperaturas	14 chars
CT	RCT	Posición + ID de conductor por iButton	17 chars (`;` + 16 hex)
CU	RCU	Posición + estado de sesión + código	2+ chars (estado + `;` + código)
BQ	RBQ	Posición + batería de respaldo	3 chars
BR	RBR	Posición + respaldo + 1 temp	10 chars
BV	RBV	Posición + respaldo + 2 temps	17 chars
HQ	RHQ	Posición + respaldo + horómetro	11 chars
HR	RHR	Posición + respaldo + horómetro + 1 temp	18 chars
HV	RHV	Posición + respaldo + horómetro + 2 temps	25 chars
EQ	REQ	Posición + datos CAN OBD-II	Variable (segmento aparte)
ER	RER	Posición + datos CAN J1939	Variable (segmento aparte)

Nota sobre CQ vs CP: CQ aplica un filtrado de calidad GPS (mínimo de satélites, máx PDOP). CP transmite el fix GPS crudo sin filtrar. Ambos comparten la misma base de 66 chars sin sufijo extra.

Formatos de sufijo extendido

CR -- 1 Temperatura (7 chars después de la base)

Offset	Largo	Campo	Formato	Conversión
0-4	5	Temp0Raw	Decimal signado `+NNNN` o `-NNNN`	**valor / 10** = grados Celsius
5-6	2	Temp0Age	**Hex** `HH`	Hex a decimal = segundos desde la última lectura

Ejemplo: `+0235` = +23.5 °C, age `0A` = hace 10 segundos

Importante: El campo de antigüedad de la temperatura es hexadecimal (0x00-0xFF = 0-255 segundos), no decimal.

CV -- 2 Temperaturas (14 chars)

Offset	Largo	Campo
0-4	5	Temp0Raw (decimal signado)
5-6	2	Temp0Age (hex)
7-11	5	Temp1Raw (decimal signado)
12-13	2	Temp1Age (hex)

CT -- iButton (17 chars)

Offset	Largo	Campo	Descripción
0	1	Separador	Carácter literal `;` (siempre presente)
1-16	16	iButton ID	ROM ID Dallas 1-Wire DS1990, 8 bytes como 16 caracteres hex

Es un identificador estándar de 64 bits Dallas/Maxim 1-Wire usado para identificación de conductor. Los 8 bytes incluyen: 1 byte de family code + 6 bytes de número de serie + 1 byte de CRC.

Importante: El sufijo CT comienza con un `;` literal antes de los 16 chars hex. Hay que saltarse el `;` inicial al extraer el ID.

CU -- Sesión (variable, 2+ chars)

Offset	Largo	Campo	Valores
0	1	Estado	`0` = cerrada (conductor desconectado), `1` = abierta (conductor conectado)
1	1	Separador	Carácter literal `;`
2+	var	SessionCode	Código de operador/conductor que identifica la sesión (alfanumérico, longitud según configuración)

Ejemplo: `1;DRIVER01` = sesión abierta con código de operador `DRIVER01`. Una "sesión" representa un período de trabajo trackeado. El dispositivo lleva tiempo acumulado, distancia y horas de motor por sesión.

BQ -- Batería de respaldo (3 chars)

Offset	Largo	Campo	Conversión
0-2	3	BackupVoltageRaw	Decimal, **valor / 100** = Volts

Ejemplo: `416` = 4.16 V. Es el voltaje de la batería interna de respaldo del dispositivo.

BR -- Respaldo + 1 Temperatura (10 chars)

Offset	Largo	Campo
0-2	3	BackupVoltageRaw
3-7	5	Temp0Raw
8-9	2	Temp0Age (hex)

BV -- Respaldo + 2 Temperaturas (17 chars)

Offset	Largo	Campo
0-2	3	BackupVoltageRaw
3-7	5	Temp0Raw
8-9	2	Temp0Age (hex)
10-14	5	Temp1Raw
15-16	2	Temp1Age (hex)

HQ -- Respaldo + Horómetro (11 chars)

Offset	Largo	Campo	Conversión
0-2	3	BackupVoltageRaw	Decimal, **valor / 100** = Volts
3-10	8	HorometerRaw	Hex a decimal = **segundos** de marcha del motor

Ejemplo: `4160003 6EE0` → respaldo `416` = 4.16 V, horómetro `00036EE0` = 0x36EE0 = 225.000 segundos = 62,5 horas.

HQ incluye el voltaje de la batería interna de respaldo del dispositivo seguido del horómetro del motor. Para horómetro con voltaje de respaldo **más** sensores de temperatura, usar HR (1 temp) o HV (2 temps).

HR -- Respaldo + Horómetro + 1 Temperatura (18 chars)

Offset	Largo	Campo
0-2	3	BackupVoltageRaw
3-10	8	HorometerRaw
11-15	5	Temp0Raw
16-17	2	Temp0Age (hex)

HV -- Respaldo + Horómetro + 2 Temperaturas (25 chars)

Offset	Largo	Campo
0-2	3	BackupVoltageRaw
3-10	8	HorometerRaw
11-15	5	Temp0Raw
16-17	2	Temp0Age (hex)
18-22	5	Temp1Raw
23-24	2	Temp1Age (hex)

EQ/ER -- Datos del bus CAN (variable)

Los datos CAN **no** son parte del sufijo posicional -- aparecen como un segmento aparte delimitado por `;` dentro del paquete. Ver Sección 15.

---

7. Reporte de versión (RVR)

Respuesta a la consulta de versión de firmware del dispositivo.

```
>RVR {VERSION_STRING};[#MSGNUM];ID=DEVICEID;*CHECKSUM<
```

Campo	Descripción
VERSION_STRING	Cadena libre que identifica firmware y variante de hardware
MSGNUM	Presente cuando responde a un comando de consulta de versión (tendrá el bit 15 seteado)

Ejemplos reales:

```
>RVR RINHO IOT v1.09.20 SM BG95 LC86G 8MB;#8042;ID=860012345678901;*CD<
>RVR RINHO IOT v1.09.20 ZE EG915U LC86G 4MB;#8042;ID=860012345678901;*AB<
>RVR RINHO IOT v1.09.20 SP BG96 GPS96 8MB;#8042;ID=860012345678901;*EF<
```

Formato del version string: `{PRODUCTO} v{VERSION} {MODELO} {MODEM} {GPS} {FLASH}`

Componente	Valores	Descripción
PRODUCTO	`RINHO IOT`	Nombre del producto
VERSION	`1.09.20`	Versión de firmware (mayor.menor.parche)
MODELO	`SM`, `ZE`, `SP`	Modelo de dispositivo: SM=Smart, ZE=Zero, SP=Spider
MODEM	`BG95`, `BG96`, `UG96`, `EG915U`	Chip de módem celular
GPS	`LC86G`, `LC86L`, `GPS96`	Módulo GPS
FLASH	`4MB`, `8MB`, `16MB`	Tamaño de memoria flash

8. Keep-Alive (KA)

Mensaje de heartbeat. **No requiere ACK**.

```
>KA;ID=DEVICEID;*CHECKSUM<
```

- Intervalo por defecto: **60 segundos** (configurable por dispositivo).
- El servidor debería actualizar la tabla NAT (UDP) o trackear la vitalidad de la conexión (TCP) al recibir un KA.
- **No** enviar ninguna respuesta a un mensaje KA.

---

9. Mensajes de inventario del dispositivo

Los dispositivos pueden reportar información de hardware. Estos suelen ser respuestas a comandos de consulta de inventario:

Prefijo	Campo	Ejemplo	Formato
`RCXHWI`	ID de hardware	`>RCXHWI2210;#8020;ID=860012345678901;*AB<`	4 dígitos: códigos de modem + GPS + ADC + LEDs
`RSN`	Número de serie	`>RSNAABBCCDDEEFF000000000042;#8021;ID=860012345678901;*CD<`	24 chars hex: dirección MAC (invertida) + padding + checksum XOR
`RIMEI`	IMEI	`>RIMEI860012345678901;#8022;ID=860012345678901;*EF<`	IMEI estándar de 15 dígitos
`RTAG`	Etiqueta de configuración	`>RTAGFLEET_01;#8023;ID=860012345678901;*GH<`	Cadena de etiqueta configurable por el usuario

Estos van a tener `msgNum >= 0x8000` porque son respuestas a comandos del servidor (QCX HWI, QSN, QIMEI, QTAG).

10. Mecanismo de ACK

Cuándo enviar ACK

- Hay que enviar ACK cuando el dispositivo manda un mensaje **con** `#MSGNUM` donde `msgNum < 0x8000`.
- **NO enviar ACK:** mensajes keep-alive (KA), mensajes con `msgNum >= 0x8000` (son respuestas a comandos), o mensajes sin `#MSGNUM`.

Formato del ACK

```
>ACK;#MSGNUM;ID=DEVICEID;*CHECKSUM<
```

El servidor repite exactamente el `MSGNUM` y `DEVICEID` del mensaje recibido.

Timing del ACK (crítico)

El ACK se debe enviar SOLO DESPUÉS de que el mensaje fue persistido/procesado correctamente. Si la persistencia falla, no enviar ACK -- el dispositivo va a retransmitir. Esto garantiza no perder datos.

Ejemplo

El dispositivo envía:

```
>RCQ28060426...;#002A;ID=860012345678901;*5E<
```

El servidor responde:

```
>ACK;#002A;ID=860012345678901;*XX<
```

Donde `XX` es el checksum XOR de `>ACK;#002A;ID=860012345678901;*`.

---

11. Comportamiento de retransmisión del dispositivo

Si el dispositivo no recibe ACK después de enviar un reporte:

Parámetro	Valor
Timeout por intento	**7 segundos**
Reintentos máximos	**4**
Ventana total antes de abandonar	~35 segundos
Tamaño de la ventana de envío	Hasta **9** mensajes en paralelo
Comportamiento al agotarse	El mensaje se descarta tras agotar todos los reintentos

El dispositivo mantiene una ventana de envío de hasta 9 mensajes pendientes. Cada mensaje se retransmite de forma independiente con su propio ciclo de timeout. Después de 4 reintentos fallidos (sin ACK), el mensaje se descarta.

Protección contra "poison": Si el dispositivo detecta 3 o más reseteos consecutivos de conexión sin ningún ACK exitoso, descarta todos los mensajes pendientes para evitar enviar datos viejos al reconectar.

Manejo del número de mensaje (msgNum)

- Rango: `0x0000` a `0x7FFF` (0 a 32767).
- Wrapping: después de `0x7FFF`, el contador vuelve a `0x0000`.
- Cada mensaje obtiene un msgNum único e incremental.
- El msgNum se usa para asociar ACKs con mensajes enviados -- el servidor debe repetir el msgNum exacto.

---

12. Envío de comandos (servidor a dispositivo)

Formato del mensaje de comando

Los comandos del servidor al dispositivo usan el mismo framing TAIP:

```
>COMMAND_BODY;#MSGNUM;ID=DEVICEID;*CHECKSUM<
```

Componente	Descripción
COMMAND_BODY	La cadena del comando (específica del dispositivo, opaca para el servidor)
MSGNUM	Asignado por el servidor, 4 dígitos hex (rango 0x0001-0x7FFF)
DEVICEID	Identificador del dispositivo destino
CHECKSUM	Checksum XOR de todo desde `>` hasta `*`

Construcción del comando

1. Elegir un `msgNum` (4 dígitos hex, rango 0x0001-0x7FFF). Incrementar por cada comando.
2. Armar el mensaje parcial: `>COMMAND;#MSGNUM;ID=DEVICEID;*`
3. Calcular el checksum XOR de esa cadena.
4. Agregar checksum + `<`.

Ejemplo -- pidiendo la versión de firmware:

```
cmd = ">QVR;#0042;ID=860012345678901;*"
checksum = calculate_checksum(cmd)  # ej: "3F"
full_cmd = cmd + checksum + "<"     # ">QVR;#0042;ID=860012345678901;*3F<"
```

Cuerpo del comando

El cuerpo del comando es **opaco** -- puede ser virtualmente cualquier cadena que el firmware del dispositivo reconozca. El dispositivo tiene más de 100 comandos internos para configuración, consultas y control. El catálogo específico depende del firmware del dispositivo y queda fuera del alcance de este documento de protocolo.

Lo que importa para la capa de ingestión es:

- **El framing** siempre es el mismo: `>BODY;#MSGNUM;ID=DEVICEID;*CS<`
- **El checksum** siempre se calcula igual.
- **La confirmación** siempre sigue la convención `msgNum | 0x8000`.

Comandos desconocidos

Si el dispositivo recibe un comando que no reconoce, **lo ignora silenciosamente** -- no se envía respuesta de error. El servidor debería implementar un timeout para detectar comandos sin confirmar.

---

13. Flujo de confirmación de comandos

Convención del número de mensaje

Rango	Dirección	Significado
`0x0000-0x7FFF`	Dispositivo a Servidor	Reportes del dispositivo (posición, versión, etc.)
`0x0000-0x7FFF`	Servidor a Dispositivo	Comandos del servidor
`0x8000-0xFFFF`	Dispositivo a Servidor	**Confirmación de un comando** (repite el msgNum del servidor con el bit 15 seteado)

Lógica de confirmación

Cuando el servidor envía un comando con `#0042`:

1. El dispositivo lo recibe y lo ejecuta.
2. El dispositivo responde con `msgNum = 0x0042 | 0x8000 = 0x8042`.
3. El cuerpo de la respuesta contiene el resultado del comando (por ej. version string, dump de configuración, o solo un eco).

Ejemplo de flujo:

Servidor envía: `>QVR;#0042;ID=860012345678901;*3F<` (consulta versión de firmware)

Dispositivo responde: `>RVR RINHO IOT v1.09.20 SM BG95 LC86G 8MB;#8042;ID=860012345678901;*AB<` (respuesta, msgNum 0x8042)

Detectando respuestas

```
msg_num_int = int(msg_num_hex, 16)
is_response = msg_num_int >= 0x8000
original_cmd_num = msg_num_int & 0x7FFF  # quitar bit 15 para obtener el original
```

14. Parámetros adicionales (AP=/PA=)

Algunos reportes incluyen un segmento extra `;AP=` o `;PA=` con datos personalizados de sensores. Ambos prefijos son equivalentes (la versión de firmware determina cuál se usa). El segmento contiene parámetros tipados separados por comas, generados por scripts de reporte configurados por el usuario.

Formato

```
>R{TIPO}{base...};AP={param1},{param2},...;#MSGNUM;ID=DEVICEID;*CHECKSUM<
>R{TIPO}{base...};PA={param1},{param2},...;#MSGNUM;ID=DEVICEID;*CHECKSUM<
```

Estructura del parámetro

Cada parámetro sigue el formato `nombre:tipo:valor`, donde:

Tipo	Significado	Formato del valor	Ejemplo
`1`	Entero	Número entero	`fuel:1:0`, `rpm:1:1496`
`2`	Float	Número decimal	`temp:2:28.9`, `hum:2:75.2`
`3`	String	Texto (sin `;` ni chars de control)	`status:3:ok`, `driver:3:Juan`

Los nombres de parámetro son alfanuméricos con guiones bajos (`[a-zA-Z0-9_]`).

Algunos scripts legacy pueden omitir el campo de tipo, produciendo pares `nombre:valor` (por ej. `c0:1496`). La capa de ingestión debería manejar ambos formatos.

Ejemplos reales

Datos CAN de un script vehicular (formato legacy sin tipos):

```
>RHQ03...;PA=id:0000059278,c0:1496,c2:61,c4:,c6:84,c7:;#B6BD;ID=863238070628120;*16<
```

Datos de sensor BLE/analógico (formato tipado):

```
>RBQ05...;AP=fuel:1:0,temp:2:28.9;#1016;ID=860012345678901;*2B<
>RCQ00...;AP=hum:2:75.2,alt:2:-15.5;ID=860012345678901;*FF<
```

Extracción

1. Buscar `;AP=` en el paquete crudo. Si no se encuentra, buscar `;PA=` .
2. Extraer desde después del `=` hasta el siguiente `;` .
3. El resultado es una cadena separada por comas con los parámetros, que se puede dividir y tipar más a fondo si hace falta.

---

15. Datos de bus CAN (EQ/ER)

OBD-II (EQ)

```
>REQ{66 chars base};{PID=VALOR,PID=VALOR,...};#MSGNUM;ID=DEVICEID;*CS<
```

J1939 (ER)

```
>RER{66 chars base};{SPN=VALOR,SPN=VALOR,...};#MSGNUM;ID=DEVICEID;*CS<
```

Los datos CAN aparecen como un segmento aparte (entre delimitadores `;`) que contiene pares `CODIGO=VALOR`, normalmente separados por comas. Un segmento puede contener un solo parámetro (sin coma) o varios.

Ejemplos:

```
>REQ...;2010=1000.00,5000=0.00;#0001;ID=860012345678901;*AB<
>REQ...;1=WVWZZZ3CZWE123456;#0002;ID=860012345678901;*CD<
```

Identificando el segmento CAN: Dividir el paquete por `;`. Saltar segmentos que coincidan con prefijos conocidos (`ID=`, `AP=`, `PA=`, `TXT=`, `#`, `*`) y segmentos que empiezan con `R` (el body del reporte). El segmento CAN es el que contiene `=`. Buscar primero segmentos con `=` y `,` (multi-parámetro), y como fallback los segmentos con solo `=` (mono-parámetro).

Nota: Los códigos PID/SPN y sus unidades son específicos del vehículo y dependen del estándar OBD-II/J1939. Este documento no incluye una tabla de lookup de PID/SPN -- consultar referencias estándar de PIDs OBD-II (por ej. SAE J1979) o bases de datos SPN J1939 para su interpretación.

---

16. Estructura de salida parseada

Después de parsear un paquete TAIP crudo, debería producirse la siguiente estructura de datos. Este esquema es agnóstico a la implementación -- usalo para alimentar tu pipeline interno, base de datos, message queue o API.

Campos comunes (todos los tipos de mensaje)

Campo	Tipo	Ejemplo	Descripción
`device_id`	string	`"869456012345678"`	Identificador del dispositivo (típicamente el IMEI)
`msg_type`	string	`"position"`	Clasificación del mensaje (ver abajo)
`msg_num`	string	`"002A"`	4 dígitos hex en mayúscula (vacío si está ausente)
`is_response`	boolean	`false`	`true` si msgNum >= 0x8000 (respuesta a comando)
`event_time`	datetime	`2026-04-06T15:30:25Z`	Timestamp GPS del dispositivo (UTC)
`received_at`	datetime	`2026-04-06T15:30:26Z`	Hora de recepción en el servidor (UTC)
`raw`	string	`">RCQ...;*5E<"`	Paquete original completo (para debug/replay)

Nota sobre timestamps: `event_time` viene del reloj GPS del dispositivo y puede tener desvío. `received_at` es el reloj del servidor. Usar `received_at` para ordenar cuando la confiabilidad del reloj del dispositivo es incierta.

Valores de msg_type

Valor	Descripción
`position`	Reporte de posición GPS (familia RCQ)
`version`	Reporte de versión de firmware (RVR)
`confirmation`	Respuesta a un comando del servidor cuando el body no coincide con un tipo conocido
`keepalive`	Heartbeat (KA)
`unknown`	Tipo de mensaje no reconocido (no es respuesta)

Prioridad de clasificación: Primero clasificar por contenido del body (position, version, keepalive). Después chequear `is_response` (msgNum >= 0x8000). Si el body coincidió con un tipo conocido, **mantener ese tipo** y poner `is_response = true`. Solo usar `confirmation` cuando `is_response = true` Y el body NO coincidió con ningún tipo conocido. Por ejemplo, una respuesta de versión `>RVR ...;#8042;...` tiene `msg_type = "version"` con `is_response = true`, no `msg_type = "confirmation"`.

Campos de posición (cuando msg_type = "position")

Campo	Tipo	Ejemplo	Descripción
`latitude`	float	`-34.62000`	Grados decimales
`longitude`	float	`-58.38000`	Grados decimales
`speed`	integer	`45`	km/h
`course`	integer	`180`	Grados (0-359)
`ignition`	boolean	`true`	Derivado del bit 7 de inputs

Campos de telemetría (de la base de 66 chars)

Campo	Ejemplo crudo	Conversión	Convertido
`report_type`	`"HR"`	--	Subtipo del reporte
`report_id`	`"28"`	--	Secuencia hex 00-FF
`inputs`	`"A2"`	Bitmask hex	Bit 7=IGN, bits 6-0=entradas digitales
`outputs`	`"03"`	Bitmask hex	Bits 0-2=salidas digitales
`voltage_raw`	`"128"`	/ 10	12.8 V
`odometer_raw`	`"00000011"`	Hex a decimal	17 metros
`gps_power`	`"1"`	--	0=Apagado, 1=Encendido
`gps_mode`	`"3"`	--	2=2D, 3=3D
`pdop`	`"05"`	--	Entero decimal, más bajo=mejor
`satellites`	`"10"`	--	Entero decimal
`gps_age`	`"0512"`	Hex a decimal	1298 segundos desde el fix
`modem_power`	`"1"`	--	0=Apagado, 1=Encendido
`gsm_status`	`"1"`	--	Estado de registro 0-5
`csq`	`"15"`	--	0-30 dBm, 99=sin señal

Campos de telemetría extendidos (presentes según el tipo de reporte)

Campo	Tipos de reporte	Ejemplo crudo	Conversión
`backup_voltage_raw`	BQ, BR, BV, HQ, HR, HV	`"416"`	/ 100 = 4.16 V (batería interna)
`horometer_raw`	HQ, HR, HV	`"00036EE0"`	Hex a decimal = segundos de marcha del motor
`temp0_raw`	CR, CV, BR, BV, HR, HV	`"+0235"`	/ 10 = +23.5 °C
`temp0_age`	CR, CV, BR, BV, HR, HV	`"0A"`	**Hex** a decimal = 10 segundos
`temp1_raw`	CV, BV, HV	`"-0253"`	/ 10 = -25.3 °C
`temp1_age`	CV, BV, HV	`"0B"`	**Hex** a decimal = 11 segundos
`ibutton`	CT	`"0102030405060708"`	ROM ID Dallas 1-Wire DS1990 (16 chars hex, el `;` inicial del crudo se elimina)
`session`	CU	`"1"`	`"0"` = cerrada, `"1"` = abierta
`session_code`	CU	`"DRIVER01"`	Código de operador/conductor que identifica la sesión activa
`can_protocol`	EQ, ER	`"OBD-II"`	OBD-II o J1939
`can_data`	EQ, ER	`"2010=1000.00"`	Pares CODIGO=VALOR separados por coma
`ap_params`	Cualquiera	`"fuel:1:0,temp:2:28.9"`	Parámetros adicionales del segmento `;AP=` o `;PA=`

Campos de versión (cuando msg_type = "version")

Campo	Tipo	Ejemplo	Descripción
`version`	string	`"RINHO IOT v1.09.20 SM BG95 LC86G 8MB"`	Descriptor completo de firmware + hardware

Campos de inventario (cuando están presentes en la respuesta)

Campo	Tipo	Ejemplo	Descripción
`hwi`	string	`"2210"`	4 dígitos: códigos de modem + GPS + ADC + LEDs
`sn`	string	`"AABBCCDDEEFF000000000042"`	24 chars hex derivados de la dirección MAC del dispositivo
`imei`	string	`"860012345678901"`	IMEI de 15 dígitos (igual a device_id)
`tag`	string	`"FLEET_01"`	Etiqueta configurable por el usuario

Ejemplo completo parseado

Entrada:

```
>RHR28060426153025-3462000-
05838000045180A203128000000110000050A10051241600036EE0+02350A;#002C;ID=860012345678901;*BC<
```

Salida:

```
{
  "device_id": "860012345678901",
  "msg_type": "position",
  "msg_num": "002C",
  "is_response": false,
  "event_time": "2026-04-06T15:30:25Z",

  "latitude": -34.62000,
  "longitude": -58.38000,
  "speed": 45,
  "course": 180,
  "ignition": true,

  "report_type": "HR",
  "inputs": "A2",
  "outputs": "03",
  "voltage_raw": "128",
  "odometer_raw": "00000011",
  "gps_power": "0",
  "gps_mode": "3",
  "pdop": "05",
  "satellites": "10",
  "gps_age": "0512",
  "modem_power": "1",
  "gsm_status": "1",
  "csq": "15",

  "backup_voltage_raw": "416",
  "horometer_raw": "00036EE0",
  "temp0_raw": "+0235",
  "temp0_age": "0A"
}
```

17. Modelos de dispositivo

Existen tres variantes de hardware: **Rinho Zero** (ZE), **Rinho Spider** (SP) y **Rinho Smart** (SM). Difieren en periféricos de hardware (modem, GPS, ADC), pero a nivel del protocolo son idénticos.

Cualquier modelo puede generar cualquier tipo de reporte. El tipo de reporte se configura por dispositivo (vía scripts o comandos), no se determina por el modelo de hardware. Un Rinho Zero puede mandar reportes RHV igual que un Rinho Smart puede mandar reportes RCQ simples. El parser **no** debería filtrar ni restringir tipos de reporte según el modelo del dispositivo.

Todos los modelos soportan:

- Todos los tipos de reporte de posición ASCII (CQ, CP, CR, CV, CT, CU, BQ, BR, BV, HQ, HR, HV, EQ, ER)
- Consulta de versión (QVR produce RVR)
- Keep-alive (KA)
- Intervalos de reporte y triggers de evento configurables
- Detección de ignición (input bit 7)
- Comandos bidireccionales con confirmación

18. Consideraciones de seguridad

El protocolo TAIP tal como lo usan los dispositivos Rinho tiene las siguientes características de seguridad:

- **Sin cifrado.** Toda la información se transmite en texto plano sobre UDP/TCP.
- **Identificación solo por ID.** No hay autenticación criptográfica entre dispositivo y servidor. Cualquier origen que conozca el ID de un dispositivo podría enviar paquetes spoofeados.
- **Contraseña a nivel de dispositivo.** Los dispositivos soportan una contraseña para comandos de configuración (SPW/QPW), pero esto es para la gestión del dispositivo, no para autenticar reportes.

Recomendación para integradores: Usar seguridad a nivel de red (VPN, whitelisting de IP, reglas de firewall) para restringir qué orígenes pueden enviar datos al puerto de ingestión. Considerar validar los IDs de dispositivo contra un registro conocido.

19. Checklist de integración

- [ ] **Listener UDP y/o TCP** en puerto configurable
- [ ] **Escaneo de frames:** encontrar todos los mensajes `>...<` delimitados en los datos recibidos
- [ ] **Buffering de stream TCP:** buffereando los bytes entrantes y extrayendo frames `>...<` completos a lo largo de los segmentos
- [ ] **Validación de checksum** (XOR de bytes desde `>` hasta `*`, comparar con los 2 dígitos hex después de `*`)
- [ ] **Parser ASCII:** extraer todos los campos de la base posicional de 66 chars
- [ ] **Todos los tipos extendidos de reporte:** CR, CV, CT, CU, BQ, BR, BV, HQ, HR, HV, EQ, ER
- [ ] **Envío de ACK** después de procesar exitosamente (no para KA, no para msgNum >= 0x8000)
- [ ] **Mantenimiento de tabla NAT** para UDP (deviceId a IP:puerto)
- [ ] **Numeración de mensajes:** detectar respuestas (msgNum >= 0x8000)
- [ ] Parseo de **RVR** (versión de firmware)
- [ ] **Mensajes de inventario del dispositivo:** RCXHWI, RSN, RIMEI, RTAG
- [ ] **Envío de comandos** con construcción de checksum
- [ ] **Matching de confirmación de comandos** (msgNum | 0x8000)
- [ ] **Extracción de datos del bus CAN** (segmentos EQ/ER)
- [ ] **Extracción de parámetros adicionales** (segmentos `;AP=` / `;PA=`)
- [ ] **Manejo de keep-alive** (refresh de NAT, sin ACK)

Errores comunes

1. **La velocidad es km/h** -- NO nudos. No multiplicar por 1.852.
2. **El divisor de latitud es 100000** -- da grados decimales (ej: -3462000 = -34.62000 grados).
3. **El divisor de longitud es 100000** -- ídem (ej: -05838000 = -58.38000 grados).
4. **PDOP es decimal** -- el campo de 2 chars es un entero decimal (0-99), no hex.
5. **La antigüedad de la temperatura es hex** -- el campo de 2 chars de age SÍ es hex (0x00-0xFF = 0-255 segundos).
6. **El odómetro es en metros** -- string hex, convertir a decimal.
7. **El voltaje está en décimas** -- `128` = 12.8V.
8. **ACK antes de persistir = pérdida de datos** -- siempre persistir primero, después ACK.
9. **MsgNum 0x8000+ es una respuesta** -- no enviar ACK, no tratar como un reporte nuevo.
10. **Múltiples paquetes por datagrama** -- siempre escanear TODOS los frames `>...<` en los datos recibidos.

Apéndice A: Mensajes de ejemplo

Posición estándar (CQ)

```
>RCQ28060426153025-3462000-05838000045180A203128000000110000050A100512;#002A;ID=860012345678901;*5E<
```

Posición con temperatura (CR)

```
>RCR28060426153025-3462000-
05838000045180A203128000000110000050A100512+02350A;#002B;ID=860012345678901;*AF<
```

Posición con respaldo + horómetro (HQ)

```
>RHQ28060426153025-3462000-
05838000045180A203128000000110000050A10051241600036EE0;#002B;ID=860012345678901;*BC<
```

Sufijo `4160003 6EE0` (sin espacio): respaldo `416` = 4.16 V + horómetro `00036EE0` = 62,5 horas.

Posición con respaldo + horómetro + 2 temps (HV)

```
>RHV28060426153025-3462000-05838000045180A203128000000110000050A10051241600036EE0+02350A-
02530B;#002C;ID=860012345678901;*BC<
```

Posición con iButton (CT)

```
>RCT28060426153025-3462000-
05838000045180A203128000000110000050A100512;01ABCDEF12345678;#002D;ID=860012345678901;*AA<
```

Notar el separador `;` literal antes de los 16 chars hex del iButton ID.

Posición con sesión (CU)

```
>RCU28060426153025-3462000-
05838000045180A203128000000110000050A1005121;DRIVER01;#002E;ID=860012345678901;*AB<
```

Estado `1` (abierta) seguido de `;` y código de operador `DRIVER01`.

Respuesta de versión

```
>RVR RINHO IOT v1.09.20 SM BG95 LC86G 8MB;#8042;ID=860012345678901;*CD<
```

Keep-Alive

```
>KA;ID=860012345678901;*2F<
```

ACK

```
>ACK;#002A;ID=860012345678901;*45<
```

Comando (servidor a dispositivo)

```
>QVR;#0042;ID=860012345678901;*3F<
```

Confirmación de comando (dispositivo a servidor)

```
>RVR RINHO IOT v1.09.20 SM BG95 LC86G 8MB;#8042;ID=860012345678901;*AB<
```

Datos CAN (OBD-II)

```
>REQ28060426153025-3462000-
05838000045180A203128000000110000050A100512;2010=1000.00,5000=0.00;#002D;ID=860012345678901;*DE<
```

Apéndice B: Referencia rápida de conversión de tipos de datos

Campo crudo	Almacenado como	Conversión	Unidad
Latitud	`+NNNNNNN` o `-NNNNNNN`	/ 100000	Grados decimales
Longitud	`+NNNNNNNN` o `-NNNNNNNN`	/ 100000	Grados decimales
Velocidad	Decimal `NNN`	Directo	km/h
Rumbo	Decimal `CCC`	Directo	Grados (0-359)
Voltaje	Decimal `JJJ`	/ 10	Volts
Voltaje de respaldo	Decimal `NNN`	/ 100	Volts
Odómetro	Hex `KKKKKKKK`	Hex a decimal	Metros
Horómetro	Hex `NNNNNNNN`	Hex a decimal	Segundos (marcha del motor)
Temperatura	Decimal signado `+NNNN`	/ 10	Grados Celsius
Antigüedad de temperatura	**Hex** `HH`	Hex a decimal	Segundos
Antigüedad de GPS	Hex `PPPP`	Hex a decimal	Segundos
PDOP	Decimal `NN`	Directo	Entero (más bajo = mejor)

Este documento describe el protocolo TAIP de Rinho v1.1. El protocolo soporta transporte UDP y TCP con framing de mensajes idéntico.