

OrbisNetLogger 解説

2026.10.12 @Tanaka

フローチャート

受信機から UBX・NMEA を取り込み、ファイルに保存するソフト OrbisNetLogger のフローチャートです。上から下へ、矢印の順に見てください。

図 1 は全体像です。データは左から右へ 1 本の道の流れ、下の 2 つのタイマーがそれを見守ります。

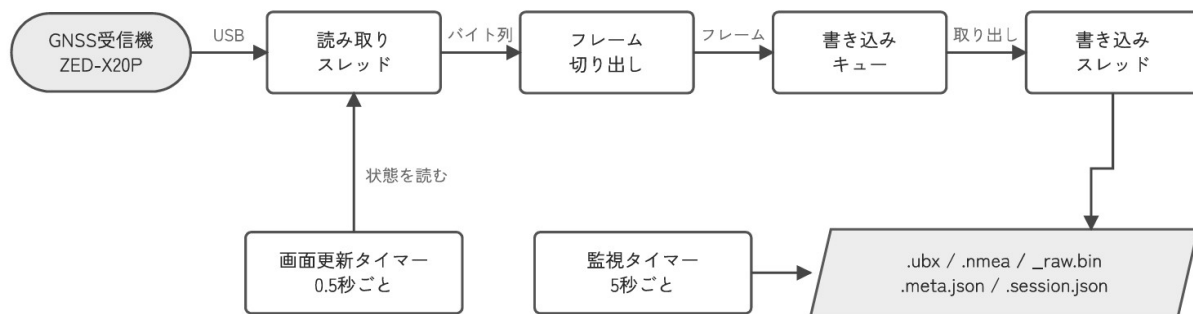


図 1 全体のしくみ

読み取りと書き込みを別スレッドにすることで、ディスクが一瞬遅くても受信を取りこぼしません。

ポイント

- ・ 受信機は USB の仮想 COM ポートとして見えるので、読み取りには SerialPort を使うだけです。
- ・ 読み取りと書き込みを別スレッド+キューでつなぐのが設計の核心です。ディスクが一瞬詰まっても、受信側は止まりません。
- ・ 画面更新(0.5 秒)と監視(5 秒)は、記録そのものとは独立した「見張り役」です。

対応するコード

RecorderService.Start / Stop	全体の起動と停止
RecorderService.ReaderLoop	読み取りスレッド
RecorderService.WriterLoop	書き込みスレッド
RecorderService.GetSnapshot	画面表示用の状態取得

図2は、「記録開始」を押してから「記録中」になるまでの手順です。失敗しうる所を先に済ませるのがコツです。

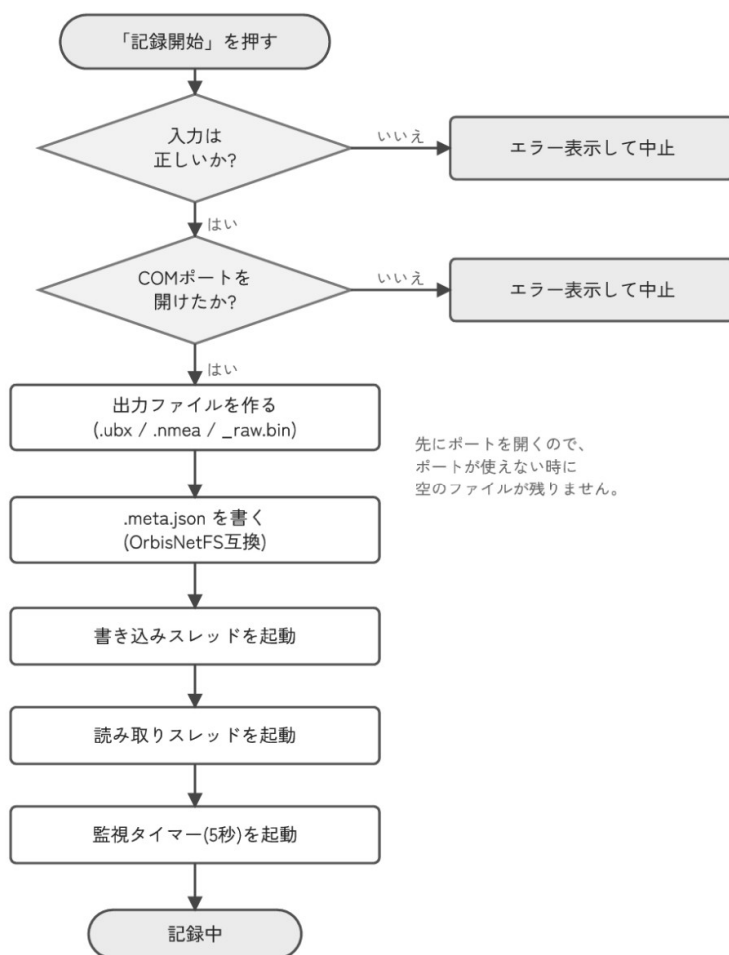


図2 記録開始

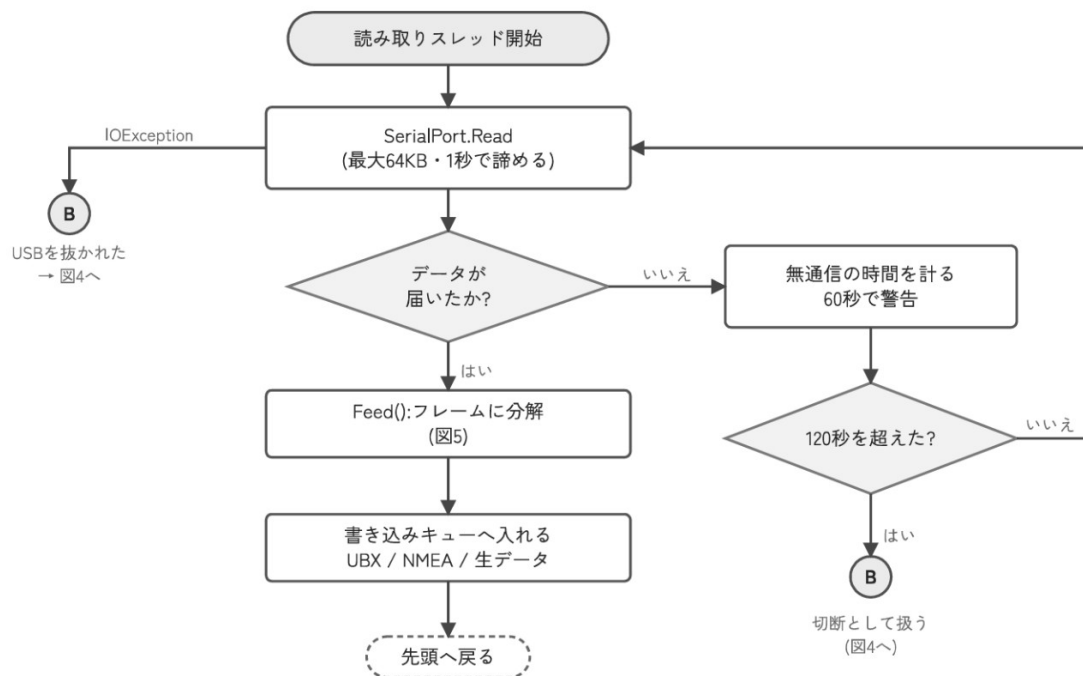
ポイント

- ・ ひし形は「はい/いいえ」で分かれる判断です。失敗したら中止します。
- ・ ポートを開く→ファイルを作る、の順にしているので、ポートが使用中でも空ファイルは残りません。
- ・ .meta.json は記録開始時に1回だけ書きます(OrbisNetSigma が読む観測情報)。

対応するコード

RecorderService.Start	手順全体
RecorderService.OpenPort	ポートを開く
SigmaMetaWriter.Write	.meta.json の出力

図3は、記録中ずっと回り続ける心臓部です。「届いた/届かない/例外」の3通りに分かれます。



先頭へ戻る = 毎回 Read() に戻る。これが記録中ずっと繰り返される。

図3 受信ループ(読み取りスレッド)

ポイント

- ・ Read は最大1秒だけ待ちます。タイムアウトは異常ではなく、「今は届いていない」という正常な結果です。
- ・ 無通信が60秒で警告、120秒で切断扱いです(秒数は画面で変更可能。30秒間隔の観測でも誤って切らないため既定は120秒)。
- ・ USBを抜くと IOException が起きるので、これも切断として図4へ進みます。

対応するコード

RecorderService.ReaderLoop	このループ本体
StreamSplitter.Feed	フレームへの分解
RecorderOptions.NoDataTimeoutSeconds	無通信タイムアウト

図4は、USBが抜けた、または長く無通信だった時の復旧手順です。成功するまで2秒おきに繰り返します。

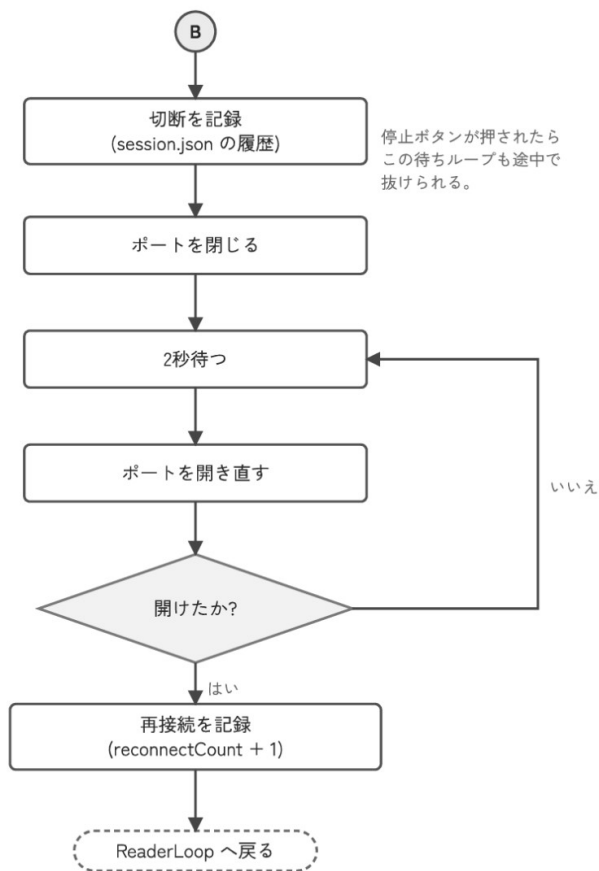


図4 切断と再接続

ポイント

- ・ 切断の事実はすぐ記録し、後で session.json の履歴から原因調査できます。
- ・ 開けなかった場合は「2秒待つ」へ戻る、という“ループ”が図の右側の矢印です。
- ・ 再接続できれば、同じファイルに続きを書き続けます(ファイルは分かれません)。

対応するコード

RecorderService.HandleDisconnect	切断の記録と再接続
DisconnectRecord	切断履歴のデータ

図5は、途切れたり混ざったりして届くバイト列から、壊れていない UBX と NMEA だけを取り出す仕組みです。

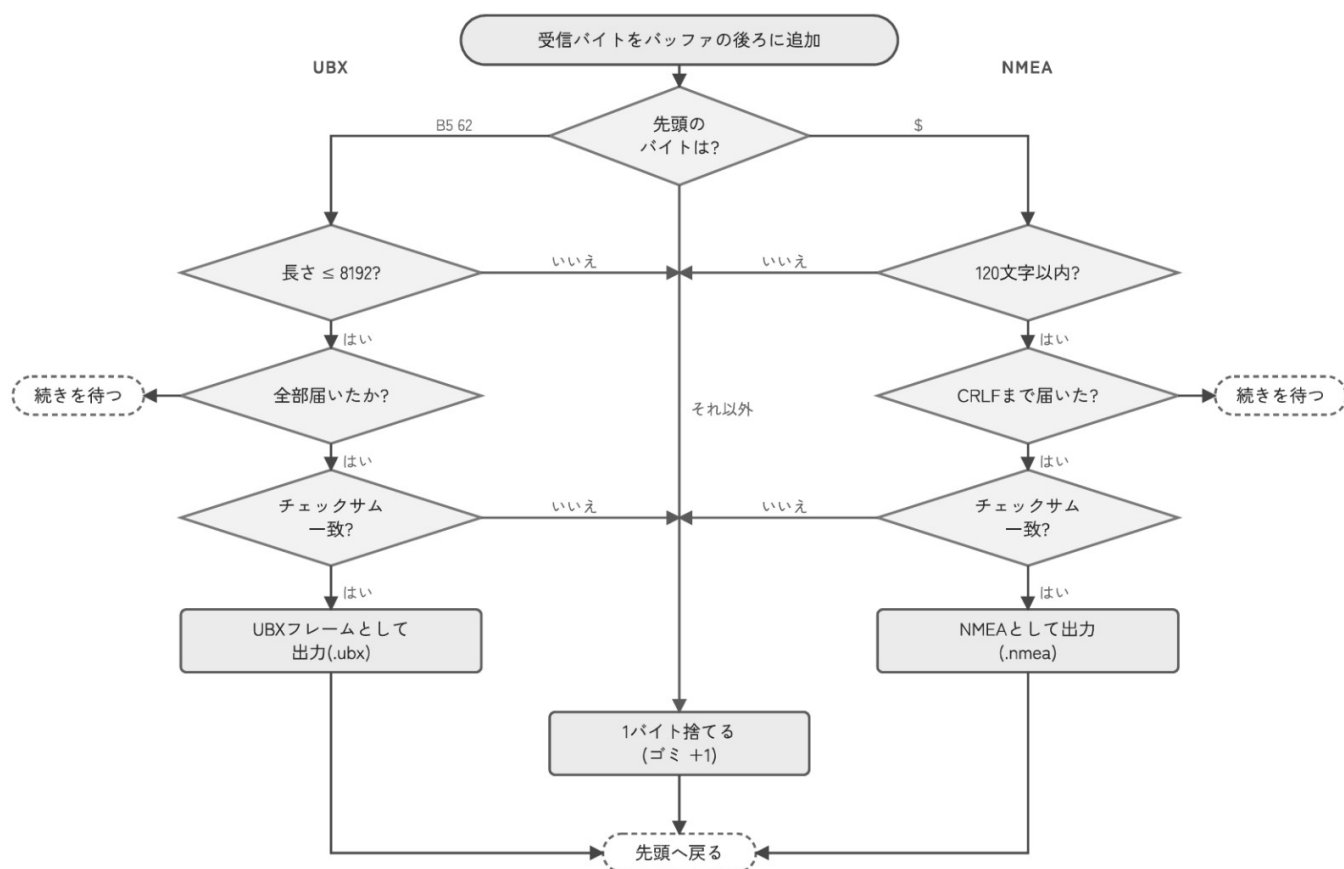


図5 フレーム切り出し

ポイント

- ・ UBX は「B5 62 + 種別 + 長さ + 本体 + 2 バイトのチェックサム」。NMEA は「\$... ¥*XX 改行」で XOR のチェックサム。
- ・ 最大長(UBX 本体 8192、NMEA 120 文字)を超えたら「ヘッダの読み違い」とみなします。
- ・ 不合格のときは 1 バイトだけ捨てて探し直します。これが「再同期」で、ゴミのバイト数として統計に出ます。

対応するコード

StreamSplitter.Process	先頭バイトで振り分け
StreamSplitter.TryUbx	UBX の検査
StreamSplitter.TryNmea / ValidateNmea	NMEA の検査

図6は、キューから取り出して、種類ごとのファイルに書き込みます。ループは「停止処理でキューが閉じられ、中身が空」で終わります。

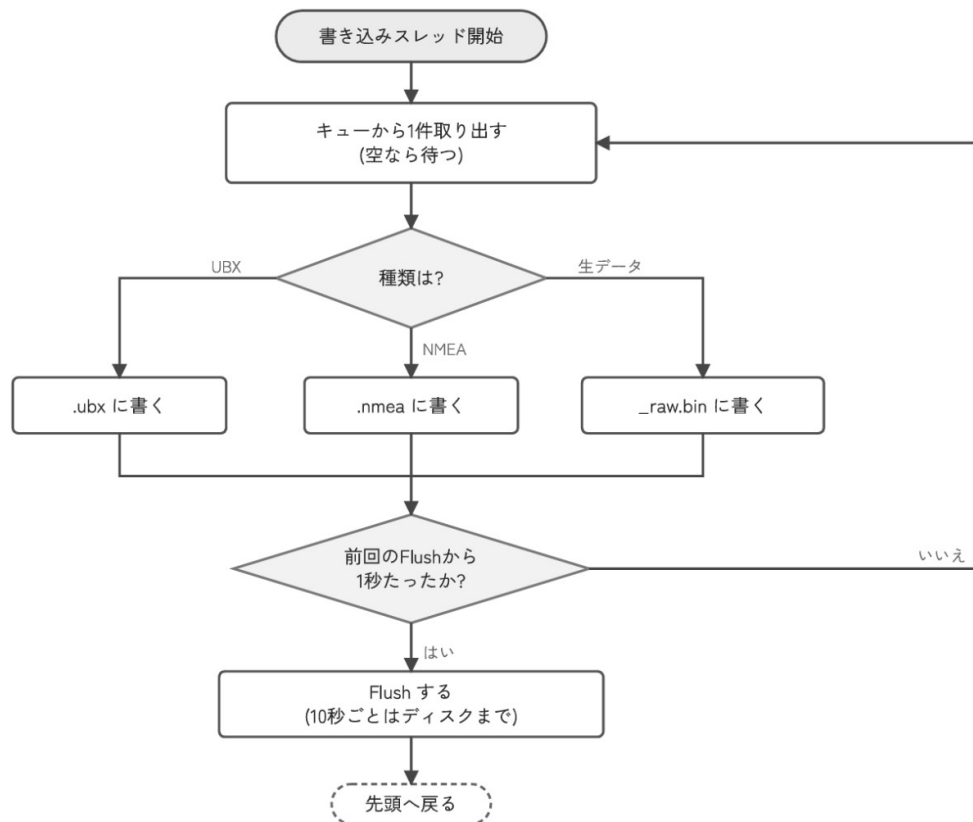


図6 書き込みスレッド

ポイント

- ・ キューが空のときは待つだけで、CPU を使いません(BlockingCollection)。
- ・ 1 秒ごとに Flush して、OS の書き込み待ちを減らします。約 10 秒ごとは物理ディスクまで書き出し、突然の電源断でも失うのは最大 10 秒分に抑えます。

対応するコード

RecorderService.WriterLoop	本体
FlushIfDue	1 秒ごとの Flush 判定

図 7 は、記録とは別に、5 秒ごとに「異常がないか」を見て回る監視タイマーです。

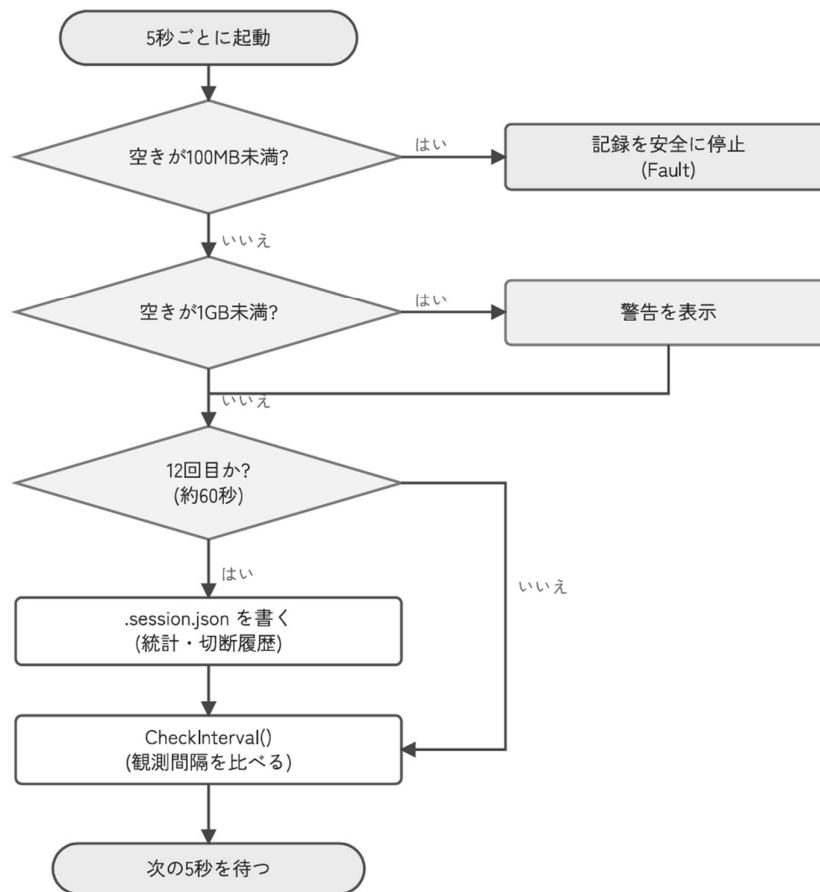


図 7 監視タイマー

ポイント

- ・ 空き容量は 2 段階。1GB 未満で警告、100MB 未満なら安全に停止します(ファイルを壊さないため)。
- ・ 12 回に 1 回(約 60 秒)だけ session.json を更新します。
- ・ CheckInterval は、入力した間隔と実測(RAWX の時刻差)が 10%以上ずれていると 1 回だけ警告します。

対応するコード

RecorderService.HousekeepingTick	本体
RecorderService.CheckInterval	観測間隔の確認
RecorderService.TrackRawxEpoch	RAWX の時刻差を測る

図8は、停止ボタンを押した時の処理です。止める時は「入口→出口」の順です。順番を間違えるとデータが欠けます。

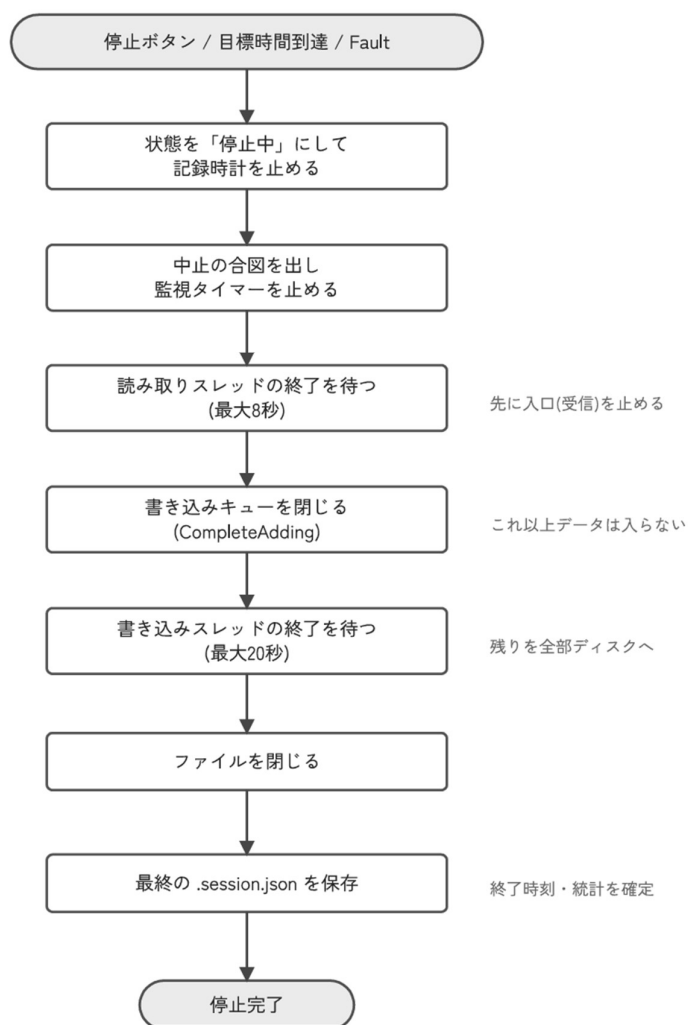


図8 停止処理

ポイント

- ・ 先に読み取りを止め、次にキューを閉じ、最後に書き込みスレッドが残りを出し切ります。
- ・ 待ち時間に上限(8秒・20秒)があるのは、万一スレッドが固まってもアプリが終了できるようにするためです。

対応するコード

RecorderService.Stop	本体
MainWindow.OnClosing	閉じる時の確認