CarLinkアプリ改修開発作業

UNIXドメインソケットによるログデータ転送機能の追加

META PROTOCOL CORPORATION

[初版] 2014/10/27

1. 追加機能について

本書では、自動車に搭載のOBD-IIコネクタを介してCANメッセージをUSBおよびBluetooth経由で収集記録できるAndroid用ドライブレコーダアプリ（以下，CarLink CANusbAccessory および CarLink CAN-BT）に対して、UNIXドメインソケットを介して同Android端末上の他のサービスアプリにログデータを逐次転送可能な機能を追加したことを説明します。

追加機能の一覧を以下に示します。

- USB接続版とBluetooth接続版の両方のCarLinkアプリに対して、UNIXドメインソケットにて、
  別アプリ（サービス）へログデータを転送する機能を追加しました。
- 転送先のUNIXドメインソケットの名前を設定できるメニュー項目を追加しました。

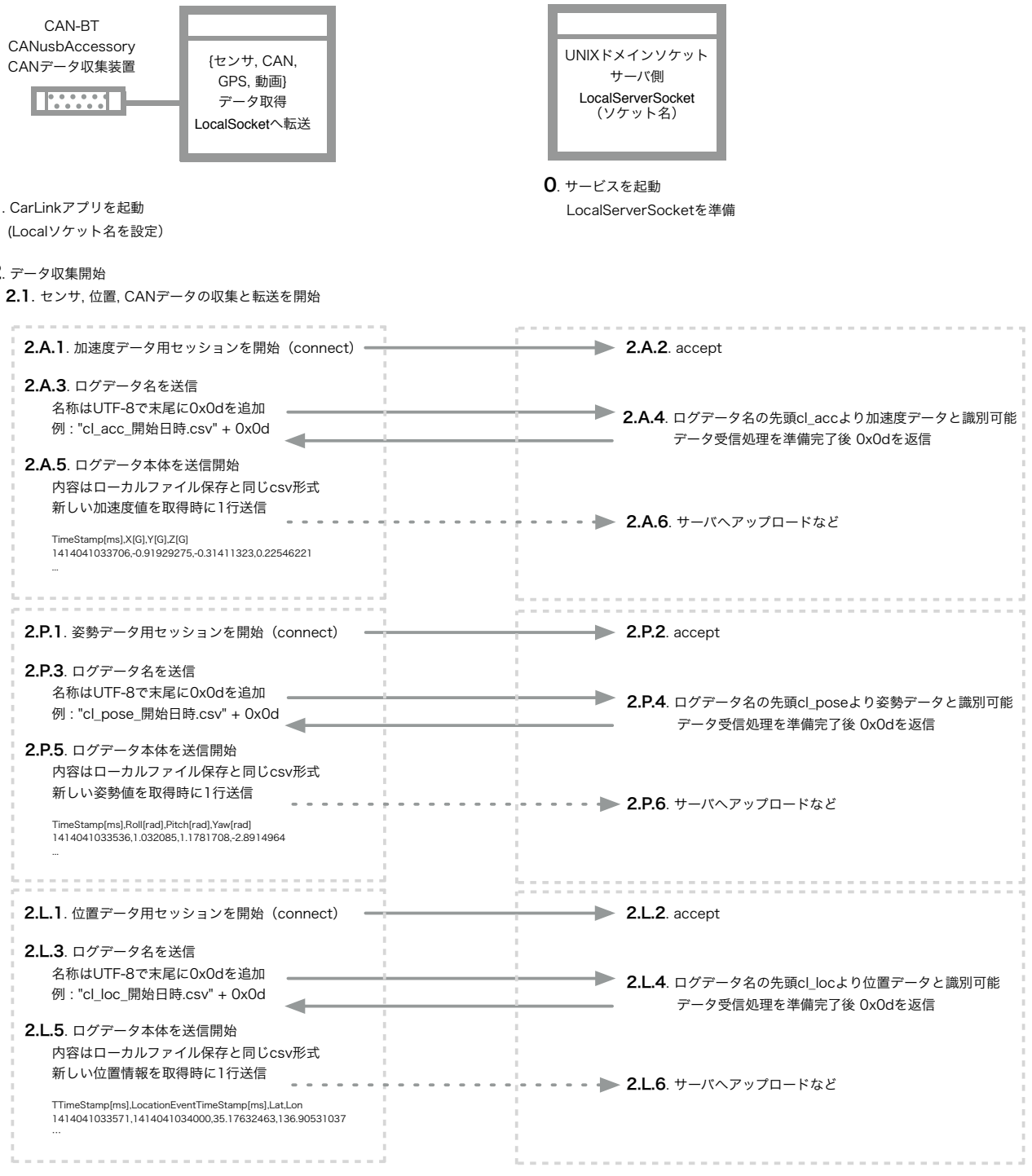
- 転送するタイミングは以下の通り。
  - 加速度、姿勢、GPS、CANデータは、新着データがあったタイミングで転送
  - 映像データ(.mp4ファイル)は、映像ファイルを閉じたタイミングで転送
  - 映像データの転送タイミングの調整のため、インターバル録画できる機能と、
    間隔を1秒から60秒まで1秒刻み、2分から60分まで1分刻みで設定できるメニュー項目を追加しました。

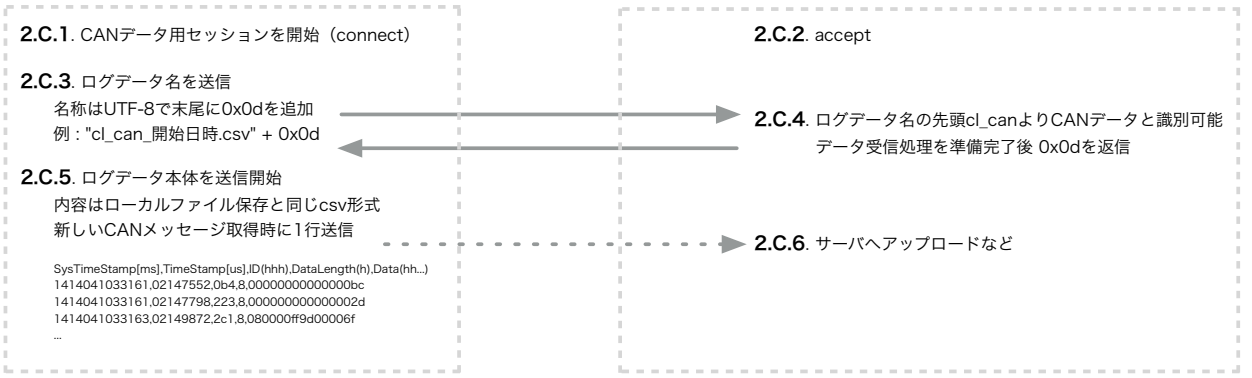
以降、2. UNIXドメインソケットを介したログデータ転送動作の詳細と、3. CarLinkアプリ側での転送機能の利用手順、4. 受信側サービスアプリのサンプルプログラムを説明します。

2. UNIXドメインソケットによるログデータ転送機能

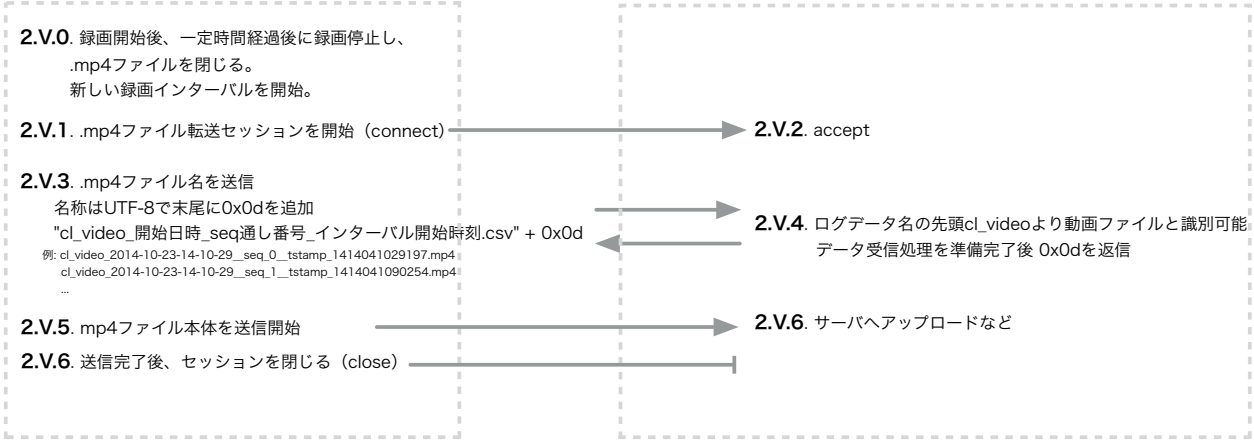
ログデータ転送動作の流れを下図に示します。CarLinkアプリ側がUNIXドメインソケットのクライアント側、受信側のサービスアプリがサーバー側ソケットになります。

先にサービスを起動し(Step.0)、その後、CarLinkアプリ側で通常の記録開始操作を行います(Step.1から2)。転送の停止は、通常の停止操作後に行われます(Step.3)。





2.2. 動画のインターバル録画を開始
（以下を一定間隔0sec～60minで繰り返し）



3. データ収集停止操作時
各種転送セッションをclose

3. 利用手順

3.1 アプリのインストール

付属CD-ROMより、Android端末に、CarLink CAN-BT / CANusbAccessory LS(LocalSocket)バージョンのAPKファイルをインストールします。

また、受信側のサービスアプリのサンプルプログラム、もしくは、作成された受信側サービスプログラムもインストールする必要があります。サンプルの受信側サービスアプリ（4節を参照）のAPKファイルは、付属CD-ROMに含まれています。

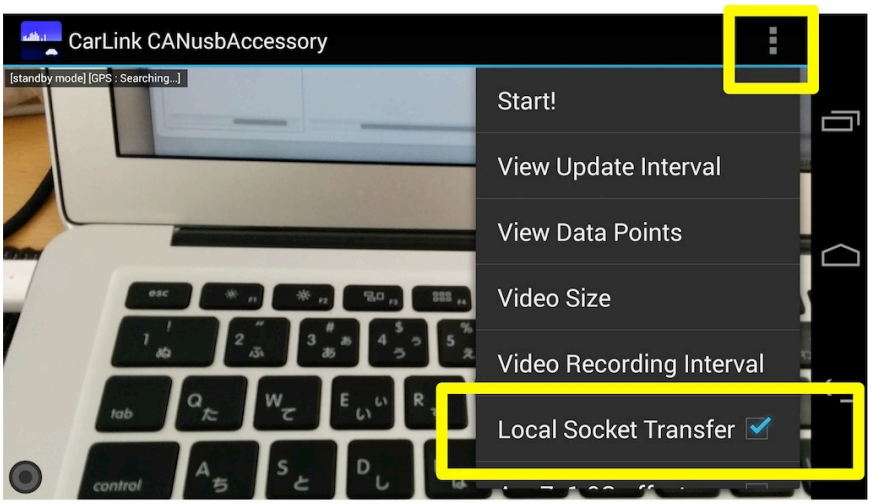
3.2 CarLinkアプリ側 転送先UNIXドメインソケット名の設定

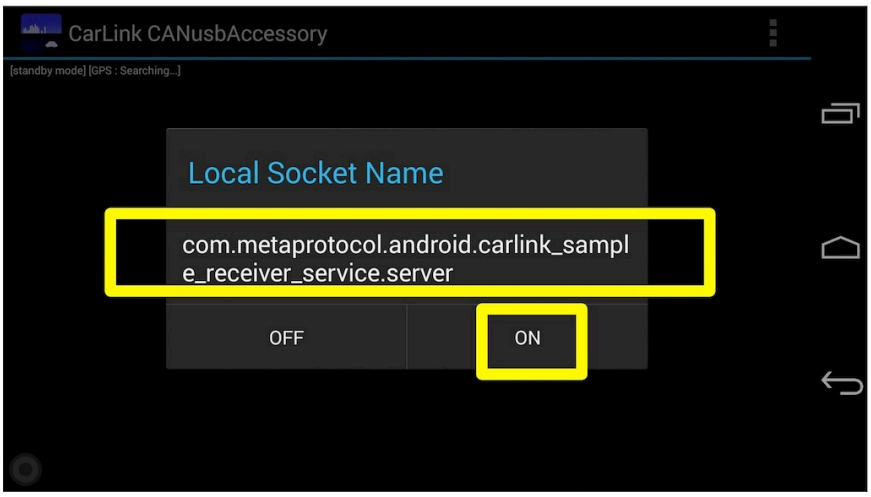
CarLinkアプリにてログデータの転送機能を有効にするには、2節にて示した転送先のUNIXドメインローカルソケット名をあらかじめ設定する必要があります。

CarLinkアプリを起動し、下図の「LocalSocketTransfer□」メニューより、ソケット名を入力できます。

メニュー項目をクリックすると入力ダイアログが表示されます。ソケット名を入力後、「ON」ボタンを押します。

メニュー項目のチェックマークは転送機能の有効状態を示します。もう一度、メニューをクリックすると無効状態に切り替えられます。

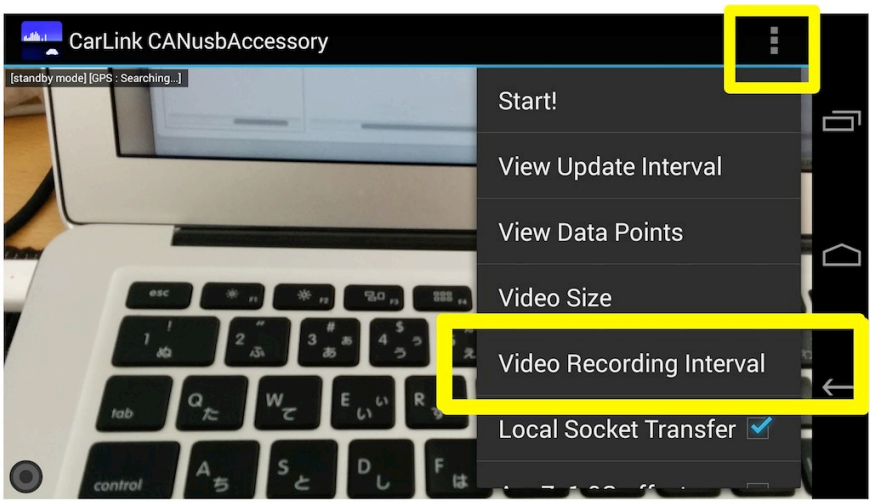


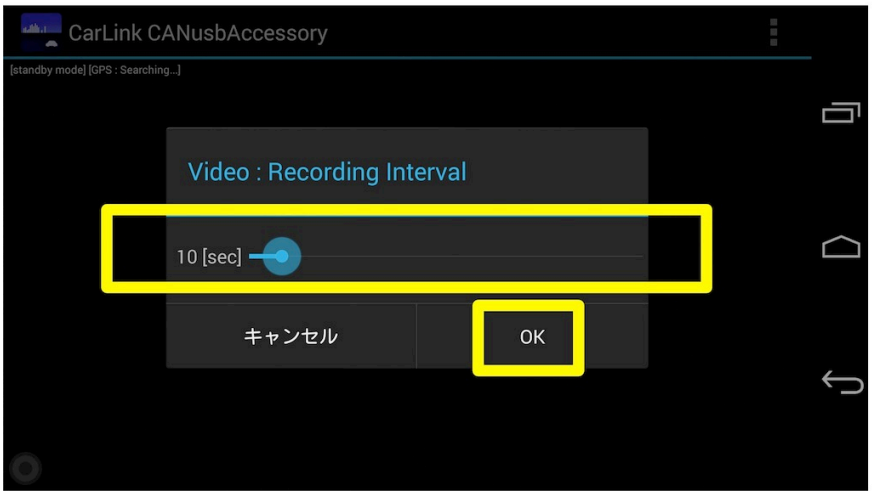


3.3 CarLinkアプリ側 映像ファイルの録画インターバルの設定

2節にて示しましたように、データ収集における映像データの録画時間を一定間隔に区切ることができます。間隔を下図のメニュー項目「Video Recording Interval」より指定できます。

間隔を0[sec]にした場合は、映像データを区切りません。1[sec]以降に指定した時に、映像データを区切ります。
1[sec]から60[sec]までは1[sec]単位、2[min]から60[min]までは1[min]単位で指定できます。





3.4 データ記録開始と転送開始

先に、受信側サービスアプリ（サンプルプログラムの場合は、SampleDataReceiverServiceLauncherアプリ）を起動しておきます。
続いて、通常のCarLinkアプリの利用手順を同様に「START」メニュー項目（もしくは画面左下の○ボタン）よりデータの記録開始を指示すると、同時に設定したUNIXドメインソケットに接続を開始します。
停止も通常通り「STOP」メニュー（もしくは画面左下の○ボタン）より指示可能です。

サンプルの受信サービスアプリでは、受信したデータを、ローカル内蔵ストレージのCarLinkSampleDataReceiverServiceフォルダ内に書き出します。

4. 受信側サービスアプリのサンプルプログラム概説

付属CD-ROMに、2節の図中の右側の受信サービスアプリに相当するサンプルプログラムとして、CarLink_SampleDataReceiverServiceを収納しています。

サンプルプログラムのファイル構成を以下に示します。

- **src/.../SampleDataRecevierService.java**
  サービス本体。UNIXドメインソケットのサーバ（LocalServerSocket）の初期化やAccept待ちスレッド、接続されたセッション毎のデータ受信スレッドを持つ。

- **src/.../SampleDataRecevierServiceLauncher.java**
  上項のサービスを起動するだけのActivity。

- **src/.../ICarLinkDataReceiverService.aidl および ICarLinkDataReceiverServiceCallback.aidl**
  ActivityとService間のインタフェース。サンプル中では特に利用なし。

サンプルプログラムでは、転送されたログデータを受診後、Android端末のローカルストレージに保存する動作が記述されています。

ファイル保存処理箇所を、ネットワーク転送するなどの別処理に置き換えてお使いください。ログデータ受信スレッドは、SampleDataRecevierServiceクラス中の以下の223行から342行までに該当します。ソケット名も受信サービスアプリの開発に合わせて変更してお使いください。

```
//-----------------------------------------------------------------
// 接続確立後のセッション用スレッド
class DataReceiveThread extends Thread
{
	private LocalSocket local_socket_;
	public boolean exit_flag_;

	private String data_name_;

	public DataReceiveThread(LocalSocket local_socket)
	{
		local_socket_ = local_socket;
		exit_flag_ = false;
	}

	public void close()
	{
		exit_flag_ = true;
		interrupt();
	}

	@Override
	public void run()
	{
```

```
Log.d(TAG, "DataReceiveThread.start() called.");

// データ受信ループバッファリングしながら、別サーバへの転送処理などを行う。
// ここではサンプル的な動作として、ファイルに書き出す。
try
{
	BufferedInputStream buffered_inp_stream = null;

	buffered_inp_stream = new BufferedInputStream(local_socket_.getInputStream());

	// --
	// ヘッダを受信
	// ログファイルの名称を、改行コード0x0dを終端として受信
	// データ種別は名称の先頭の数文字で判断可能。cl_acc_...、cl_pose_...、cl_loc_...、cl_video_など。
	data_name_ = "---";
	{
	byte[] header_bytes = new byte[512];

	int i = 0;
	while( (!(Thread.currentThread().isInterrupted())) && (!exit_flag_) )
	{
		int val = buffered_inp_stream.read();
		if(val == -1) { exit_flag_ = true; break; }
		else if(val == 0x0d) { break; }

		header_bytes[i] = (byte)val;

		i++;
		// 長過ぎる場合
		if(i >= header_bytes.length) { exit_flag_ = true; break; }
	}

	// データ名称をStringに変換する。
	if(!exit_flag_)
	{
		data_name_ = new String(header_bytes, 0, i, "UTF-8");

		Log.d(TAG, "new DataReceiveSession(data_name = " + data_name_ + ")");
	}
	}


	// --
	// ログデータを受信
	// 以下の例ではBufferedInputStreamで読んでいますが、
	// cl_video（バイナリ）以外は、新着データが1行単位で転送されるため、
	// cl_can_..、cl_acc_などでは、必要であればBufferedReaderを使ってください。
	BufferedOutputStream buffered_out_stream
		= new BufferedOutputStream(new FileOutputStream(
			new File(received_data_outp_folder_, data_name_)));
	{
	byte[] bytes = new byte[1024 * 4];

	// --
	// ここで、データ名称の接頭文字列より種別を判定したり、
	// さらに他に転送するセッションを初期化などして、
	// 準備ができたら、0x0dを送信する。
	if(!exit_flag_)
	{
		// 開始を遅延させるテスト
		//try { Thread.sleep(4000); }catch(InterruptedException ie) { ie.printStackTrace(); }

		local_socket_.getOutputStream().write(0x0d);

		Log.d(TAG, "send 0x0d and start receiving.. (data_name = " + data_name_ + ")");
	}


	// 返信後、転送開始。
	// 以降はackなしで転送されますので、 必要であれば別の転送スレッドのキューに
```

```
                        // 逐次受信データを渡すなど、読み込みが滞らないように注意してください。
                        int read_bytes;
                        while( (!(Thread.currentThread().isInterrupted()))
                                        && (!exit_flag_)
                                        && ((read_bytes = buffered_inp_stream.read(bytes)) != -1) )
                        {
                                // 受信したデータを使って何かする。
                                // この例では、ファイルに書き出し。
                                buffered_out_stream.write(bytes, 0, read_bytes);
                        }
                        }
                        buffered_out_stream.close();

                        // --
                        // LocalSocketを閉じる。
                        buffered_inp_stream.close();
                local_socket_.close();

                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
                finally
                {
                        Log.d(TAG, "DataReceiveThread(data_name = " + data_name_ + ") exited.");
                        data_receiver_threads_.remove(this);
                }
                }
        }
```