Harry’s LapTimer

Documentation v1.15

Introduction and Scope

This paper is part of LapTimer's documentation. It covers all available editions LapTimer comes in – both for iOS and Android. In case functionality or wording differs, the document marks the respective sections using an iOS or an Android 🤖 icon. For historical reasons, most snapshots are iOS pictures. However, as both lines of apps converge over time and will show only minor differences, pictures are not doubled in general.

For further documentation http://www.gps-laptimer.de/documentation is the first address for everything.

Although the aspects described in this document go far beyond the depth and detail you will see for an app, it misses other areas completely. So far now, please consider it as a series of technical papers.

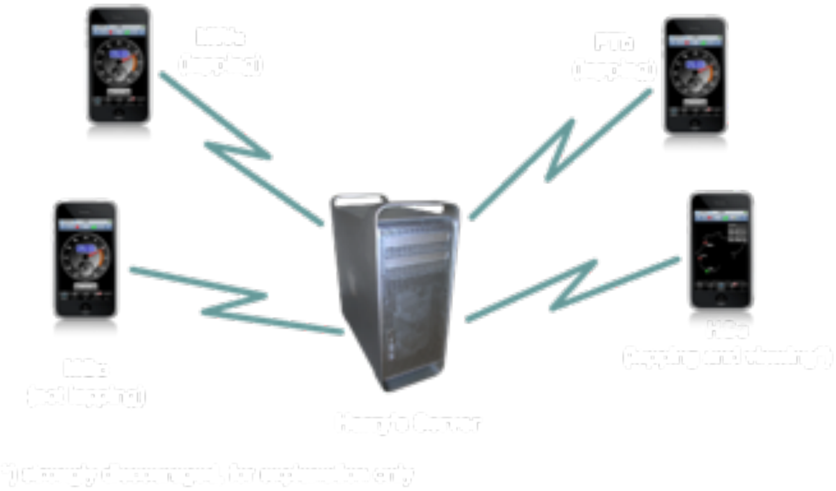
Server Integration

This section is a collection of information and conversations on the client / server protocol used by Harry’s GPS Suite, namely the LapTimer family. In addition to this, it includes porting consideration in case the server is re-implemented.

Intro

Online racing, introduced in versions 8.2 / 9.0, is an exciting feature that allows you to see your buddies on track, online, at any time.

The concept behind is a so called client / server model. Every LapTimer application is a client sending both the current position and times lapped to a dedicated server (let‘s name this server ‚Harry‘s Server‘). Harry‘s Server collects all this information and allows access to this data by LapTimer versions featuring the Online View.



LapTimer server communicating with LapTimer clients

Harry‘s LapTimer Petrolhead and above include the Online View.

So what is required for a perfect track day experience?

- An iOS/Android smartphone loaded with LapTimer Rookie (or higher) app for each driver.
- At least one smartphone / tablet with LapTimer Petrolhead (or higher).
- Mobile network around the track, all LapTimers configured to publish positions (see below).

Recommended Options:

- Power supply and / or a supported external GPS

All LapTimer editions are equipped with an alert feature broadcasting a “danger ahead” alarm to LapTimer users racing the same track:

- While lapping, LapTimer offers an Alert button.
- When pressed, the current position is broadcasted to all drivers currently connected to Harry‘s Server driving on the same track as the driver issuing the alert.
- When the position is approached by any of these drivers, an alert with a yellow flag is coming up.

The broadcast is not guarantied to reach the drivers on track:

In fact, this function is not bullet proof as several things need to work that have no defined service level: the alert needs to reach Harry's Server, thus the LapTimer instance sending, needs to have a connection to the Internet, and Harry's Server needs to be up and running (and be connected to the net - which is not always the case).

Furthermore the drivers on track need to be online too: only LapTimer clients currently connected and participating the Local Track Community group will receive the broadcast.

Message Protocol

After establishing a socket based connection, LapTimer clients and server exchange messages. This messages are binary objects derived directly from C / C++ struct memory layout. This is the reason we show C struct definitions when describing individual message formats below. Before we dive into this, let us have a look in the general structure of messages.

0	*16/32*	*variable*
`Header`	`Payload`	

Overall message layout

A message starts with a header followed by message payload. Every message comes with a message type defining the overall message size and the individual payload memory layout. As an example, a message reporting the position of a driver on track has the message type `MessageCurrentPositionV2`. To report the position of a driver, this message's payload is coming with a latitude, a longitude and a list of track IDs / groups this position should be associated with.

But before going into individual messages, let us have a look into the header.

0	*16*	*32*
`Header v1`	`Header v2 extension`	

Message header are either 16 or 32 bytes long

Due to the evolution of LapTimer's functionality, there are two types of headers used. Almost all message types use the version 2 now, but downward compatibility is keep for older message types. The type of header is defined by the message type. New message types will use the v2 header while old message types use v1 only. Consequently, the payload of new message types start in positin 32 of the message while it starts at byte 16 for old message types. Header type v2 is an extension of v1, i.e. it comes with the same 16 bytes in the beginning but adds an additional field for bytes 16 to 31.

0	*4*	*8*	*12*	*16*
`creatorID`	`sUDID`	`size`	`type`	

16	*32*
`UDID`	

Standard structure of v1 / v2 message headers

Header v1 is coming with 4 fields with 4 bytes each. `creatorID` is a constant sent in the beginning to allow receivers to both check for message validity and detect the beginning of a message in case it got out of sync. `sUDID` is the so called simplified UDID (unique device ID). It is an identifer for the device sending the message. It is a non-unique hashed value of the unique `UDID` we will see for the v2 header version. `size` the overall length of the message including both header and payload bytes. The last field of v1 headers we finally get is `type`, the message type discussed before.

Header v2 adds a single field to v1 headers, the unique `UDID` of the device sending the message. We will come back to UDIDs later, they are actually generated by the server and exchanged as a token between clients and servers to address the endpoints of communication.

To fully understand this structure, we walk through the C side of our sample. Message types are enumerated in the `enum GPSMessageType`. Our message has the message type `MessageCurrentPositionV2` mapped to `int` 44:

```
typedef
enum
{
    ...

    MessageCurrentPositionV2 = 44,                 //   Client > Server

    ...

}   GPSMessageType;
```

The message definiton is using structure inheritence, i.e. a C++ feature. We show those relevant to understand or sample. The full list of message primitives and structures can be found in this document’s appendix.

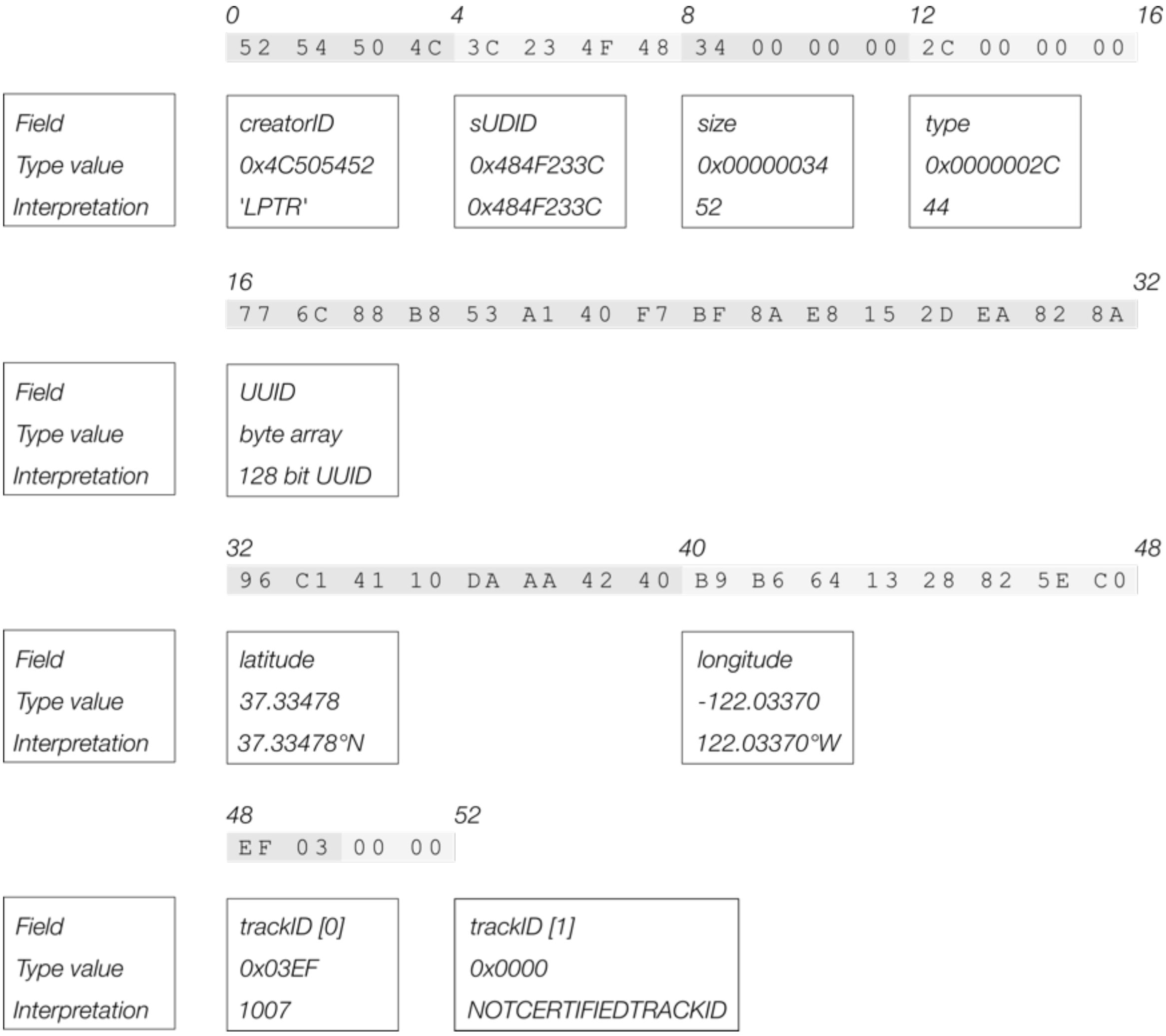
```
struct GPSClientServerMessage
{
    UInt32         lapTimerCreatorID;             //   0:
    UInt32         sUDID;                         //   4:
    UInt32         messageSize;                   //   8:
    GPSMessageType messageType;                   //   12:
                                                  //   16:
};

struct GPSClientServerV2Message : GPSClientServerMessage
{
    UUID128        UDID;                          //   16:
                                                  //   32:
};

struct GPSClientServerCurrentPositionV2Message : GPSClientServerV2Message
{
    double         latitude;                      //   32:  Position
    double         longitude;                     //   40:

    UInt16         groupAndTrackIDs [0];          //   48: NOTCERTIFIEDTRACKID
                                                  //   terminated list

                                                  //   48: Variable size
};
```

The index added as a comment shows the offset from the start of message for each field. Field types used here are `UInt16`, `UInt32`, `UUID128`, `double`. They are sent over the net using little endianess. `UInt16` and `UInt32` are unsigned integers of length 16 and 32 bits respectively. So their byte length is 2 and 4 bytes. `UUID128` is a 128 bit field, i.e. a byte array with a length of 32. `double` is a 64 bit IEEE 754 double float value with 8 bytes length. A server programmed in C /

C++ and using a little endian processor can use the C structs to “decode” incoming messages. Others will need to apply appropate transformation. For more information in endianess, see http://en.wikipedia.org/wiki/Endianness.

Sample Message

The sample message’s payload is one with a variable size. This is because it will come with one track ID (see track IDs defined for each certified track in LapTimer ▸ Administration ▸ Add-ons) and an arbitrary number of group IDs the user is a member of. For the picture below, we have just a track ID plus an array terminator.



```
0           4           8           12          16
52 54 50 4C 3C 23 4F 48 34 00 00 00 2C 00 00 00
```

Field	creatorID	sUDID	size	type
Type value	0x4C505452	0x484F233C	0x00000034	0x0000002C
Interpretation	'LPTR'	0x484F233C	52	44

```
16                                              32
77 6C 88 B8 53 A1 40 F7 BF 8A E8 15 2D EA 82 8A
```

Field	UUID
Type value	byte array
Interpretation	128 bit UUID

```
32                      40                      48
96 C1 41 10 DA AA 42 40 B9 B6 64 13 28 82 5E C0
```

Field	latitude	longitude
Type value	37.33478	-122.03370
Interpretation	37.33478°N	122.03370°W

```
48          52
EF 03 00 00
```

Field	trackID [0]	trackID [1]
Type value	0x03EF	0x0000
Interpretation	1007	NOTCERTIFIEDTRACKID

Sample message reporting a coordinate for track #1007

Message Recognition

For the LapTimer instance receiving a sequence if messages, it is important split and combine incoming data chunks into individual messages. As an example, the above 52 bytes message may arrive as a network package of 52 bytes, as part of a netzwork package of size 1024 bytes, or as two packages sized 32 and 20 bytes. To recognize messages, use the `creatorID` and the message `size`. In case you detect an inconsistency (e.g. if you expect the next message but do not get the creatorID), skip everything until you find the `0x52 0x54 0x50 0x4C` sequence again.

Furthermore, both the server and client side may receive messages from later versions of the app supporting new message types. So in case you receive a message with an unknown or unsupported message type, simply drop it. LapTimer senders are prepared to receive no reply on message and will handle this situation properly.

Communication Pattern

As mentioned above, a LapTimer server implementation is not required to implement the full set of functionality (message types) clients may use. However, there is kind of a minimal set which is either mandatory or simply makes sense to support for standard operation. Such a minimal set is listed here and should be implemented first when setting up a new LapTimer server.

Requesting a certified UUID (mandatory)

Early versions of LapTimer used a device’s unique UUID (depends on platform) and generated a simplified UUID from it. This has been done on the client side by simple generating a hash code for this private UUID. The background for this is that once a socket session ends, a LapTimer server cannot identify the client for the next session. So similar to a cookie for web browsers, the simplified UUID served as an anonymized token passed between server and client across connections. Sample applications are that clients should be allowed to delete lap times submitted to hall of fame - but only in case it is a lap submitted by this client. Or the owner of a group should be able to delete the group.

Over time, we saw that this simple and uncoordinated approach generated conflicts between different users. Hash codes of unique UUIDs are not unique… Although this did not result in any severe issues, we wanted to find a better solution keeping privacy at a similar level like that we had using the simplified UUID.

So current LapTimer servers need to provide a service to generate a unique UUID for a client contacting it the first time. And this is how it works in pseudo code:

```
LapTimer Client wants to send a message MessageXY requiring a header v2 type
    if has a certified UUID already
        uses this certifed UUID for message sending
    else
        sends message MessageRequestCertifiedUDID to server
LapTimer Server receives MessageRequestCertifiedUDID
    generates a new unique UUID and returns it by sending a MessageCertifiedUDID message
LapTimer Clients receives MessageCertifiedUDID
    stores the UUID received as certified UUID for future request
    sends message MessageXY  it wanted to send in the beginning
```

This approach is making sure every UUID is unique for clients talking to a server. In terms of privacy, LapTimer has no insights into a physical or logic UUID of the users device.

Reporting positions on track and laps finished (recommended)

All LapTimer clients lapping currently on a certified track set, send a continuous stream of `MessageCurrentPositionV2` messages to the server (see sample above). In addition, once a lap has been completed, a hall of fame entry is send by the client using message type `MessageTimeLappedCertifiedV3`. Both messages are one way only, there is no reply required by the server. The server needs to store both sets of information so it is able to handle messages `MessageRegisterGroupsAndTracks` and `MessageRequestHallOfFameCertified` correctly (see next pattern). In addition, there should be some expiration handling for positions submitted. Once a client hasn’t send positions for some hours, the server should wipe all corresponding data entries.

Users can opt out from this reporting by leaving the Local Track Community group.

Request streaming of position changes (recommended)

A user may want to display other driver’s positions for a certain track or see the position of other members of a group he/she is part of. To signal this, it will send a message of type `MessageRegisterGroupsAndTracks`. The server should memorize this request and send messages of type `MessagePositionsV2`, `MessageServerStatus` and `MessageTimesLapped`. Clients will either opt out from this stream notifications be sending a message of type `MessageRegisterGroupsAndTracks` with an empty list of tracks and groups, or simple disconnect. Both status changes need to be reflected on the server side.

Request a Hall of Fame (recommended)

A user may want to see the hall of fame for a certain track. The client will send a message of type `MessageRequestHallOfFameCertifiedV3` with many filter options. The server needs to provide the result set using a message of type `MessageHallOfFameCertified`.

Report an accident (optional)

LapTimer clients allow the user to submit the position of an accident using the button Alert. To report the position of the accident, the message type `MessageAlertOnTrack` is sent to the server. The server should forward this message (using its own UUID) to all clients registered for the track ID (see `MessageRegisterGroupsAndTracks`) the accident has been reported for. LapTimer clients receiving this broadcasted `MessageAlertOnTrack` will display a yellow flag once it approaches the position of accident.

Other services (optional)

There are a number of other functional groups around track management, submission and distribition of challenges, providing track shapes, certification and distribution of vehicle definitions, platform notification support, and group maintenance. In case you feel you want to use these, please contact us.

Appendix

Server Integration

GPSMessagePrimitives

```
//
//  GPSMessagePrimitives.h
//  HarrysGPSSuite
//
//  Created by Harald on 18.10.09.
//  Copyright 2009 Harald Schlangmann. All rights reserved.
//

#ifndef __GPSMESSAGEPRIMITIVES_H__
#define __GPSMESSAGEPRIMITIVES_H__

#include "GPSLibraryBase.h"

#include "model/UserManager.h"
#include "utility/UUID128.h"

#define UDIDSIZE                                (8+1+4+1+4+1+4+1+12+1)

/******************************************************************************************
 *
 *  Message Types just to identify all messages sent between client and server
 *  > Numbers assigned must not be changed to not break compatibility to older
 *    client versions
 *  > Deprecated messages receive basic support on server side but are removed
 *    from client side when replaced
 *
 *  UPDATE GPSMessageTypeString () when enumeration is changed!
 *
 ******************************************************************************************/

typedef
enum
{
    MessageNoMessage = -1,
    MessageCurrentPosition = 0,                 //  deprecated, Client > Server
    MessageCurrentPositionV2 = 44,              //  Client > Server, replaced MessageCurrentPosition

    MessageTimeLapped = 1,                      //  deprecated, Client > Server
    MessageTimeLappedCertified = 9,             //  deprecated, Client > Server, replaced MessageTimeLapped
    MessageTimeLappedCertifiedV2 = 12,          //  deprecated, Client > Server, replaced MessageTimeLappedCertified
    MessageTimeLappedCertifiedV3 = 35,          //  Client > Server, replaced MessageTimeLappedCertifiedV2
    MessageDeleteTimeLapped = 24,               //  deprecated, Client > Server

    MessageRegisterForTrack = 2,                //  deprecated, Client > Server
    MessageRegisterGroupsAndTracks = 45,        //  Client > Server, replaced MessageRegisterForTrack
    MessagePositions = 3,                       //  deprecated, Server > Client (continous replies to MessageRegisterForTrack)
    MessagePositionsV2 = 46,                    //  Server > Client (continous replies to MessageRegisterGroupsAndTracks)
    MessageTimesLapped = 4,                     //  Server > Client (continous replies to MessageRegisterForTrack)
    MessageServerStatus = 5,                    //  Server > Client (continous replies to MessageRegisterForTrack)

    MessageAlertOnTrack = 6,                    //  Client > Server (submitting alert) AND Server > Client (broadcasted)

    MessageRequestHallOfFame = 7,               //  deprecated, Client > Server
    MessageHallOfFame = 8,                      //  deprecated, Server > Client (reply to MessageRequestHallOfFame)
    MessageRequestHallOfFameCertified = 10,     //  deprecated, Client > Server, replaced MessageRequestHallOfFame
    MessageHallOfFameCertified = 11,            //  Server > Client
    MessageRequestHallOfFameCertifiedV2 = 23,   //  deprecated, Client > Server, replaced MessageRequestHallOfFameCertified
    MessageRequestHallOfFameCertifiedV3 = 53,   //  Client > Server, replaced MessageRequestHallOfFameCertifiedV2

    MessageRequestTracks = 13,                  //  Client > Server
    MessageTracks = 14,                         //  Server > Client (reply to MessageRequestTracks)

    MessageRequestTrackShape = 15,              //  Client > Server
    MessageTrackShape = 16,                     //  Server > Client (reply to MessageRequestTrackShape)

    MessageSubmitChallenge = 17,                //  deprecated, Client > Server
    MessageSubmitChallengeV2 = 36,              //  Client > Server, replaced MessageSubmitChallenge

    MessageRequestChallenges = 18,              //  Client > Server
    MessageChallenges = 19,                     //  Server > Client (reply to MessageRequestChallenges)

    MessageRequestChallenge = 20,               //  Client > Server
    MessageChallenge = 21,                      //  Server > Client (reply to MessageRequestCallenge)
    MessageDeleteChallenge = 22,                //  Client > Server

    MessageSubmitVehicleCertification = 30,     //  deprecated, Client > Server
    MessageSubmitVehicleCertificationV2 = 33,   //  Client > Server, replaced MessageSubmitVehicleCertification
    MessageVehicleCertification = 31,           //  Server > Client (reply to MessageSubmitVehicleCertification)
    MessageRequestVehicleFieldCompletions = 49, //  Client > Server
    MessageVehicleFieldCompletions = 50,        //  Server > Client (reply to MessageRequestVehicleFieldCompletions)
    MessageRequestVehicle = 51,                 //  Client > Server
    MessageVehicle = 52,                        //  Server > Client (reply to MessageRequestVehicle)

    MessageRegisterDevice = 25,                 //  deprecated, Client > Server
    MessageRegisterDeviceV2 = 34,               //  Client > Server, replaced MessageRegisterDevice
    MessageNotificationRead = 26,               //  Client > Server
    MessageReadNotification = 27,               //  Client > Server
    MessageNotification = 28,                   //  Server > Client (reply to MessageReadNotification)
    MessageAnyNotification = 29,                //  Server > Client (reply to MessageRegisterDevice)

    MessageRequestCertifiedUDID = 32,           //  Client > Server
    MessageCertifiedUDID = 37,                  //  Server > Client (reply to MessageRequestCertifiedUDID)

    MessageUserCredentials = 38,                //  Client > Server
    MessageUserGroupMembership = 39,            //  Client > Server

    MessageRequestGroupDetails = 40,            //  Client > Server
    MessageGroupDetails = 41,                   //  Server > Client (reply to MessageRequestGroupDetails)

    MessageRequestGroupList = 42,               //  Client > Server
    MessageGroupList = 43,                      //  Server > Client (reply to MessageRequestGroupList)

    MessageRequestUserDetails = 47,             //  Client > Server
    MessageUserDetails = 48                     //  Server > Client (reply to MessageRequestUserDetails)

}   GPSMessageType;
#define NUMMESSAGETYPES                     54  //  Number of values in GPSMessageType

//  First GPSMessageType introducing an extended header with certified UDID
#define MESSAGETYPEHASV2HEADER(MESSAGETYPE)     ((MESSAGETYPE)>=MessageRequestCertifiedUDID&&(MESSAGETYPE)!=MessageRequestHallOfFameCertifiedV3)

extern const char *GPSMessageTypeName (GPSMessageType messageType);

typedef
enum
{
    VehicleCompletionFieldNone,
    VehicleCompletionFieldMake,
    VehicleCompletionFieldModel,
    VehicleCompletionFieldStyle,
    VehicleCompletionFieldYear,
    VehicleCompletionFieldCountry,
    VehicleCompletionFieldVehicleID

}   VehicleCompletionFieldType;

#define NUMLASTLAPTIMES                     10      //  Number of laps used in MessageTimesLapped
#define DEFAULTHALLOFFAMELIMIT              200     //  Number of laps used in MessageHallOfFame*
#define MAXLIMITRESULTSETS                  2000    //  Hard number applied to all hall of fame queries

#ifdef NEEDSSTRUCTPACK
#pragma pack(push,2)
#endif

typedef
struct
{
    double          latitude;
    double          longitude;

}   Coordinate2D;

typedef
struct
{
    char            driverID [DRIVERIDLENGTH];      //  0:
    UInt32          sUDID;                          //  4:
    double          latitude;                       //  8:
    double          longitude;                      //  16:
                                                    //  24:
}   DriverPositionType;

typedef
struct
{
    UUID128         UDID;                           //  0:  no sUDID here because receivers are always UUID aware clients
    double          latitude;                       //  16
    double          longitude;                      //  24
    char            name [0];                       //  32: variable length string for name (either real name or nickname)

}   DriverPositionV2Type;
#define DRIVERPOSITIONV2SIZE(NAME) (OffsetOf (DriverPositionV2Type, name)+StrLen (NAME)+1)

typedef
struct
```

```
{
    char            driverID [DRIVERIDLENGTH];      //  0:
    UInt32          sUDID;                          //  4:
    UInt32          lapTime100;                     //  8:
                                                    //  12:
}   DriverLapTimeType;

typedef
struct
{
    char            driverID [DRIVERIDLENGTH];      //  0:
    UInt32          sUDID;                          //  4:
    UInt32          lapTime100;                     //  8:
    UUID128         UDID;                           //  12: Full UDID
                                                    //  28
}   DriverLapTimeV2Type;

typedef
struct
{
    char            driverID [DRIVERIDLENGTH];      //  0:
    UInt32          sUDID;                          //  4:
    UInt32          lapTime100;                     //  8:
    UInt32          seconds;                        //  12:
    char            marshaledVehicle [1];           //  16:
                                                    //  18:
}   DriverLapTimeDatedType;                         //  Variable size, vehicle has at least 1 byte!

typedef
struct
{
    char            driverID [DRIVERIDLENGTH];      //  0:
    UInt32          sUDID;                          //  4:
    UInt32          lapTime100;                     //  8:
    UInt32          seconds;                        //  12:
    UInt32          overallDistance10;              //  16:
    char            marshaledVehicle [1];           //  20:
                                                    //  22:
}   DriverLapTimeDatedCertifiedType;                //  Variable size, vehicle has at least 1 byte!

typedef
struct
{
    UInt16          numDrivers;                     //  0:
    UInt16          trackID;                        //  2:
    Coordinate2D    position;                       //  4:
    Boolean         hasShape;                       //  20: Packed to byte alignment because Boolean is a char too
    char            trackname [1];                  //  21:
                                                    //  22:
}   TrackType;                                      //  Variable size, trackname hat at least 1 byte!

typedef
struct
{
    UInt16          trackID;                        //  0:
    UInt16          numDownloads;                   //  2:
    UInt32          lapTime100;                     //  4:
    UInt32          submitterSUDID;                 //  8:
    UInt32          challengeCode;                  //  12:
    Boolean         listed;                         //  16:
    char            fullnameAndVehicle [0];         //  17: Variable size, fullnameAndVehicle are two zero
                                                    //  terminated c strings (both UTF8)
                                                    //  18:
}   ChallengeDescriptionType;

typedef
struct
{
    UInt16          groupID;                        //  0:
    UInt16          appCategory;                    //  2:
    UInt32          listCode;                       //  4: 0 means always listed
    Boolean         isOwner;                        //  8:
                                                    //  10:
}   GroupDefinitionType;                            //  Fixed size

#ifdef NEEDSSTRUCTPACK
#pragma pack(pop)
#endif

#endif
```

GPSMessageStructures

```
//
//  GPSMessageStructures.h
//  HarrysGPSSuite
//
//  Created by Harald on 21.07.13.
//  Copyright (c) 2013 Harald Schlangmann. All rights reserved.
//

#ifndef __GPSMESSAGESTRUCTURES_H__
#define __GPSMESSAGESTRUCTURES_H__

#include "GPSLibraryBase.h"

#include "utility/UUID128.h"
#include "model/GPSMessagePrimitives.h"
#include "model/Vehicles.h"
#include "model/PositionSets.h"

#define MINTIMEFORHALLOFFAME                (30*100)                //  30 seconds
#define MAXTIMEFORHALLOFFAME                (30*60*100)             //  30 minutes

#pragma mark Data Structures sent across the net

#pragma pack(push,2)

/*************************************************************************
 *
 *  Base structures
 *
 *************************************************************************/

struct GPSClientServerMessage
{
    UInt32                          lapTimerCreatorID;              //  0: Constant to identify sender
    UInt32                          sUDID;                          //  4: Hashed value
    UInt32                          messageSize;                    //  8: Length of message including header
    GPSMessageType                  messageType;                    //  12: Selector for message type of specializations
                                                                    //  16: Enumeration is 64 bits for 64 bit architectures...

    void setHeader (UInt32 sUDID, UInt32 messageSize, GPSMessageType messageType);
};

//  Header used by all messages with MESSAGETYPEHASV2HEADER (type)
struct GPSClientServerV2Message : GPSClientServerMessage
{
    UUID128                         UDID;                           //  16: 128 bit / 16 bytes UUID format
                                                                    //  32:

    void setHeader (UInt32 sUDID, UUID128 UUID, UInt32 messageSize, GPSMessageType messageType);
};

/*************************************************************************
 *
 *  Lapping structures
 *
 *************************************************************************/

struct GPSClientServerCurrentPositionMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverPositionType              currentPosition;                //  18:
                                                                    //  42: Fixed size
};

struct GPSClientServerCurrentPositionV2Message : GPSClientServerV2Message
{
    double                          latitude;                       //  32: Position
    double                          longitude;                      //  40:

    UInt16                          groupAndTrackIDs [0];           //  48: NOTCERTIFIEDTRACKID terminated list
                                                                    //  48: Variable size
};

struct GPSClientServerTimeLappedMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverLapTimeType               timeLapped;                     //  18:

    char                            marshaledVehicle [0];           //  30: Optional
                                                                    //  30: Variable size, not sent any more in current version
};

struct GPSClientServerTimeLappedCertifiedMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverLapTimeType               timeLapped;                     //  18:
    UInt32                          overallDistance10;              //  30: Distance recorded, used for certification

    char                            marshaledVehicle [1];           //  34: Exists with variable length or '\0'
                                                                    //  36: Variable size, not sent any more in current version
};
```

```
struct GPSClientServerTimeLappedCertifiedV2Message : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          lapEndSecondsUTC;               //  18: UTC time, 0 means use server system time
    DriverLapTimeType               timeLapped;                     //  22:
    UInt32                          overallDistance10;              //  34: Distance recorded, used for certification

    char                            marshaledVehicle [1];           //  38: Exists with variable length or '\0'
                                                                    //  40: Variable size
};

struct GPSClientServerTimeLappedCertifiedV3Message : GPSClientServerV2Message
{
    UInt16                          trackID;                        //  32: Unique track id
    UInt32                          lapEndSecondsUTC;               //  34: UTC time, 0 means use server system time
    DriverLapTimeV2Type             timeLapped;                     //  38:
    UInt32                          overallDistance10;              //  50: Distance recorded, used for certification

    char                            marshaledVehicle [1];           //  54: Exists with variable length or '\0'
                                                                    //  56: Variable size
};

struct GPSClientServerDeleteTimeLappedMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          lapEndSeconds;                  //  18: Local time just like delivered by hall of fame
    DriverLapTimeType               timeLapped;                     //  22:
                                                                    //  34: Fixed size
};

/***********************************************************************************************
 *
 *  Challenge structures
 *
 ***********************************************************************************************/


struct GPSClientServerSubmitChallengeMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          challengeCode;                  //  18: Unique code for this lap
    Boolean                         listed;                         //  22: Publically visible (listed) or not listed
    UInt32                          lapTime100;                     //  24: Lap time of challenge

    UInt8                           data [0];                       //  28: Zero terminated UTF8 encoded realname,
                                                                    //  followed by zero terminated UTF8 encoded vehiclename (mar-
shalled),
                                                                    //  followed by a compress lap representation (up to message end)
                                                                    //  28: Variable size
};

struct GPSClientServerSubmitChallengeV2Message : GPSClientServerV2Message
{
    UInt16                          trackID;                        //  32: Unique track id
    UInt32                          challengeCode;                  //  34: Unique code for this lap
    Boolean                         listed;                         //  38: Publically visible (listed) or not listed
    UInt32                          lapTime100;                     //  40: Lap time of challenge

    UInt8                           data [0];                       //  44: Zero terminated UTF8 encoded realname,
                                                                    //  followed by zero terminated UTF8 encoded vehiclename (mar-
shalled),
                                                                    //  followed by a compress lap representation (up to message end)
                                                                    //  44: Variable size
};

struct GPSClientServerDeleteChallengeMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          challengeCode;                  //  18: Unique code for this lap
                                                                    //  22: Fixed size
};

struct GPSClientServerRequestChallengeMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          challengeCode;                  //  18: Unique code for this lap
                                                                    //  22: Fixed size

    static GPSClientServerRequestChallengeMessage
        *create (UInt32 sUDID, UInt16 trackID, UInt32 challengeCode);
};

/***********************************************************************************************
 *
 *  Track monitoring structures
 *
 ***********************************************************************************************/

struct GPSClientServerRequestTracksMessage : GPSClientServerMessage
{
    Boolean                         activeTracksOnly;               //  16: Select if all tracks are submitted, or only those with
more than 0 drivers
                                                                    //  18:
};

struct GPSClientServerRegisterForTrackMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
                                                                    //  18: Fixed size
};

typedef GPSClientServerRegisterForTrackMessage GPSClientServerRequestHallOfFameForTrackMessage;
typedef GPSClientServerRegisterForTrackMessage GPSClientServerRequestHallOfFameCertifiedForTrackMessage;
typedef GPSClientServerRegisterForTrackMessage GPSClientServerRequestTrackShapeMessage;

struct GPSClientServerRequestHallOfFameCertifiedV2Message : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt16                          limit;                          //  18: Limit reply to a certain number of entries
    Boolean                         verifiedTime;                   //  20: Filter times with inplausible lap times
    Boolean                         verifiedDistance;               //  21: Filter times with wrong overallDistance
    Boolean                         namedDriversOnly;               //  22: Filter times with missing driver name
    Boolean                         sortByLapTime;                  //  23: Sorting order best first or last first
                                                                    //  24: Fixed size
};

struct GPSClientServerRequestHallOfFameCertifiedV3Message : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt16                          limit;                          //  18: Limit reply to a certain number of entries
    UInt16                          groupIDFilter;                  //  20: Filter times to members of the group id;
                                                                    //      INVALIDGROUPID is the wild card
    Boolean                         verifiedTime;                   //  22: Filter times with inplausible lap times
    Boolean                         verifiedDistance;               //  23: Filter times with wrong overallDistance
    Boolean                         namedDriversOnly;               //  24: Filter times with missing driver name
    Boolean                         sortByLapTime;                  //  25: Sorting order best first or last first
                                                                    //  26: Fixed size
};

struct GPSClientServerPositionsMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverPositionType              positions [0];                  //  18:
                                                                    //  18: Variable size
};

struct GPSClientServerPositionsV2Message : GPSClientServerV2Message
{
    UInt16                          groupOrTrackID;                 //  32: Unique group or track id
    UInt16                          numPositions;                   //  34: Number of positions following
    DriverPositionV2Type            positions [0];                  //  36: This array's elements come with variable length
                                                                    //  36: Variable size
};

struct GPSClientServerTimesLappedMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverLapTimeType               timesLapped [NUMLASTLAPTIMES+1];//  18: Might be less!!!
                                                                    //  150: Variable size
};

struct GPSClientServerHallOfFameMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverLapTimeDatedType          timesLapped;                    //  18: Packed format including variable length records...
                                                                    //  36: Variable size
};

struct GPSClientServerTracksMessage : GPSClientServerMessage
{
    UInt16                          numTracks;                      //  16:
    TrackType                       tracks [0];                     //  18: Packed format including variable length records...
                                                                    //  18: Variable size
};

struct GPSClientServerHallOfFameCertifiedMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    DriverLapTimeDatedCertifiedType timesLapped;                    //  18: Packed format including variable length records...
                                                                    //  40: Variable size
};

struct GPSClientServerTrackShapeMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt16                          numPositions;                   //  18:
    Coordinate2D                    positions [0];                  //  20: Variable size

};

struct GPSClientServerServerStatusMessage : GPSClientServerMessage
{
    char                            serverStatus [0];               //  16:
                                                                    //  16: Variable size
};

struct GPSClientServerAlertOnTrackMessage : GPSClientServerMessage
```

```
{
    UInt16                          trackID;                        //  16: Unique track id
    Coordinate2D                    pos;                            //  18:
    char                            alertMessage [0];               //  34:
                                                                    //  34: Variable size
};


/************************************************************************************************
 *
 *  Challenge structures
 *
 ************************************************************************************************/

struct GPSClientServerRequestChallengesMessage : GPSClientServerMessage
{
    UInt16                          trackIDs [0];                   //  16: List of trackIds terminated by NOTCERTIFIEDTRACKID
                                                                    //  16: Variable size
};

struct GPSClientServerChallengesMessage : GPSClientServerMessage
{
    ChallengeDescriptionType        challenge;                      //  16: Packed format including variable length records...
                                                                    //  34: Variable size
};

struct GPSClientServerChallengeMessage : GPSClientServerMessage
{
    UInt16                          trackID;                        //  16: Unique track id
    UInt32                          challengeCode;                  //  18: Unique code for this lap

    UInt8                           hlptrz [0];                     //  22: Compressed lap representation (up to message end)
                                                                    //  22: Variable size
};


/************************************************************************************************
 *
 *  Vehicle structures
 *
 ************************************************************************************************/

struct GPSClientServerSubmitVehicleCertificationMessage : GPSClientServerMessage
{
    UInt16                          vehicleID;                      //  16: Existing vehicleID
    Boolean                         originalContributor;            //  18: Existing originalContributor
    UInt16                          vehicleIndex;                   //  20: Used as a token passed for reply verification
                                                                    //      (see GPSClientServerVehicleCertificationMessage)

    UInt8                           data [0];                       //  22: Zero terminated UTF8 encoded fullname,
                                                                    //  followed by zero terminated UTF8 encoded email,
                                                                    //  followed by a compressed vehicle representation (up to mes-
sage end)
                                                                    //  22: Variable size
};

struct GPSClientServerSubmitVehicleCertificationV2Message : GPSClientServerV2Message
{
    UInt16                          vehicleID;                      //  32: Existing vehicleID
    Boolean                         originalContributor;            //  34: Existing originalContributor
    UInt16                          vehicleIndex;                   //  36: Used as a token passed for reply verification
                                                                    //      (see GPSClientServerVehicleCertificationMessage)

    UInt8                           data [0];                       //  38: Zero terminated UTF8 encoded fullname,
                                                                    //     followed by zero terminated UTF8 encoded email,
                                                                    //     followed by a compressed vehicle representation
                                                                    //     (up to message end)
                                                                    //  38: Variable size
};

struct GPSClientServerVehicleCertificationMessage : GPSClientServerMessage
{
    UInt16                          vehicleID;                      //  16: Unique vehicle ID generated for submission on server side
    Boolean                         originalContributor;            //  18: Qualify if the vehicle ID is owned by submitter or is a
                                                                    //      refinement
    UInt16                          vehicleIndex;                   //  20: Used as a token passed during request
                                                                    //  22: Fixed size
};

struct GPSClientServerRequestVehicleFieldCompletionsMessage : GPSClientServerV2Message
{
    //  All of these are considered zero terminated values; the first empty field will define
    //  the completions returned by GPSClientServerVehicleFieldCompletionsMessage; if e.g.
    //  make is "Ford" and model is empty, all Ford models are returned
    VehicleCompletionFieldType      completionsField:8;             //  32: The field we are looking for
    char                            padding [1];                    //  33: padding
    VehiclesType                    vehiclesType:16;                //  34:
    char                            make [CARTYPEMAXLENGTH];        //  36: Car make as used by Edmunds (e.g. Porsche)
    char                            model [CARTYPEMAXLENGTH];       //  84: Car model as used by Edmunds (e.g. 911)
    char                            style [CARTYPEMAXLENGTH];       //  132: Car style as used by Edmunds
    char                            year [5];                       //  180: Car model year as used by Edmunds (e.g. 2009)
    char                            country
                                        [ISOCOUNTRYCODE2LENGTH+1];  //  183: Two letter ISO country code
                                                                    //  188: Fixed size
    static GPSClientServerRequestVehicleFieldCompletionsMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID, VehicleCompletionFieldType completionsField,
                VehiclesType vehiclesType, const char *make,  const char *model, const char *style,
                const char *year, const char *country);
};

struct GPSClientServerVehicleFieldCompletionsMessage : GPSClientServerV2Message
{
    VehicleCompletionFieldType      completionsField:8;             //  32: The field the following completions are valid for
    UInt16                          numVehicles;                    //  34: Overall number of vehicles matching search criteria
    UInt16                          numCompletions;                 //  34: Number of completions available for completionsField
    char                            completions [0];                //  36: List of numCompletions values, zero terminated each
                                                                    //  36: Variable size
};

struct GPSClientServerRequestVehicleMessage : GPSClientServerV2Message
{
    UInt16                          vehicleID;                      //  32: Unique vehicle ID we request data for
                                                                    //  34: Fixed size

    static GPSClientServerRequestVehicleMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID, UInt16 vehicleID);
};

struct GPSClientServerVehicleMessage : GPSClientServerV2Message
{
    UInt16                          vehicleID;                      //  32: Unique vehicle ID data is following for
    UInt8                           hvehz [0];                      //  34: Compressed vehicle representation (up to message end)
                                                                    //  34: Variable size
};


/************************************************************************************************
 *
 *  Notification structures
 *
 ************************************************************************************************/

#define IOSDEVICETOKENSIZE          32

struct GPSClientServerRegisterDeviceMessage : GPSClientServerMessage
{
    UInt16                          trackIDs [MAXTRACKS];           //  16: List of trackIDs loaded by the users
    char                            countryCode                     //  48: Country code like "US"
                                        [ISOCOUNTRYCODE2LENGTH+1];

    char                            padding                         //  51: padding
                                        [8-(ISOCOUNTRYCODE2LENGTH+1)];

#define RESETNOTIFICATIONS          (1<<0)                          //  not yet implemented
#define SANDBOXMODE                 (1<<1)                          //  not yet implemented
    UInt8                           flags;                          //  56: Ored combination of the defines above

    UInt8                           platform;                       //  57: Device platform, IOS, ANDROIDNDK, etc

    UInt16                          sku;                            //  58: The probably upgraded SKU - used for selection of
                                                                   //      notifications
    UInt16                          baseSKU;                        //  60: The SKU from AppStore's PoV - used to select certificates

    UInt16                          deviceTokenSize;                //  62: Size of app specific token
    Byte                            deviceToken [0];                //  64: App specific token
                                                                    //  64: Variable size
};

struct GPSClientServerRegisterDeviceV2Message : GPSClientServerV2Message
{
    UInt16                          trackIDs [MAXTRACKS];           //  32: List of trackIDs loaded by the users
    char                            countryCode                     //  64: Country code like "US"
                                        [ISOCOUNTRYCODE2LENGTH+1];
    char                            padding                         //  67: padding
                                        [8-(ISOCOUNTRYCODE2LENGTH+1)];

#define RESETNOTIFICATIONS          (1<<0)                          //  not yet implemented
#define SANDBOXMODE                 (1<<1)                          //  not yet implemented
    UInt8                           flags;                          //  72: Ored combination of the defines above

    UInt8                           platform;                       //  73: Device platform, IOS, ANDROIDNDK, etc

    UInt16                          sku;                            //  74: The probably upgraded SKU - used for selection of
                                                                   //      notifications
    UInt16                          baseSKU;                        //  76: The SKU from AppStore's PoV - used to select certificates

    UInt16                          deviceTokenSize;                //  78: Size of app specific token
    Byte                            deviceToken [0];                //  80: App specific token
                                                                    //  80: Variable size
};

struct GPSClientServerNotificationReadMessage : GPSClientServerMessage
{
    UInt32                          notificationID;                 //  16: Identifies the notification that has been read

    UInt16                          deviceTokenSize;                //  20: Size of app specific token
    Byte                            deviceToken [0];                //  22: App specific token
                                                                    //  22: Variable size
};
```

```
struct GPSClientServerReadNotificationMessage : GPSClientServerMessage
{
    UInt32                          notificationID;                 //  16: Identifies the notification that has been read
                                                                    //  20: Fixed size
};

struct GPSClientServerNotificationMessage : GPSClientServerMessage
{
    UInt32                          notificationID;                 //  16: Identifies the following message

    UInt8                           data [0];                       //  20: Zero terminated message and action strings
                                                                    //  20: Variable size
    static GPSClientServerNotificationMessage *
        create (UInt32 sUDID, UInt32 notificationID,
                const char *messageText, const char *action);
};

struct GPSClientServerAnyNotificationMessage : GPSClientServerMessage
{
    char                            jsonDefinition [0];             //  16: Zero terminated string including a json dictionary UTF8
                                                                    //  16: Variable size
};

/***************************************************************************************************
 *
 *  UDID certification structures
 *
 ***************************************************************************************************/

struct GPSClientServerRequestCertifiedUDIDMessage : GPSClientServerV2Message
{
                                                                    //  Full UDID proposed by client (or certified UDID when sent
                                                                    //  from server to client, see
                                                                    //  GPSClientServerCertifiedUDIDMessage)
                                                                    //  32: Fixed size
};

struct GPSClientServerCertifiedUDIDMessage : GPSClientServerV2Message
{
                                                                    //  Full UDID proposed by client (or certified UDID when sent
                                                                    //  from server to client, see
                                                                    //  GPSClientServerCertifiedUDIDMessage)
    UUID128                         certifiedUDID;                  //  32: Certified value returned
                                                                    //  48: Fixed size
};


/***************************************************************************************************
 *
 *  User and group structures
 *
 ***************************************************************************************************/

struct GPSClientServerRequestGroupListMessage : GPSClientServerV2Message
{
    static GPSClientServerRequestGroupListMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID);
};

struct GPSClientServerUserCredentialsMessage : GPSClientServerV2Message
{
    UInt16                          sku;                            //  32: App sku used currently

    UInt32                          iconSize;                       //  34: Size of icon data starting data
    char                            data [0];                       //  38: Data for icon followed by zero terminated UTF8
                                                                    //      encoded realname,
                                                                    //  followed by zero terminated UTF8 encoded email, followed by
                                                                    //  a zero terminated user status, followed by a zero
                                                                    //  terminated UTF8
                                                                    //  vehicle description
                                                                    //  38: Variable size
    static GPSClientServerUserCredentialsMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID, UInt16 sku,
                UInt32 iconSize, Byte *iconData,
                const char *userRealnameUTF8, const char *userEMailUTF8, const char *userStatusUTF8, const char *vehicleNameUTF8);
};

struct GPSClientServerUserGroupMembershipMessage : GPSClientServerV2Message
{
    UInt16                          groupOrTrackID;                 //  32: Unique group id (special range in trackIDs)
    PositionSetStatusType           status;                         //  34: Message used for both addition / maintenance and deletion

    char                            data [0];                       //  38: Zero terminated UTF8 encoded nickname for this group
                                                                    //  38: Variable size
    static GPSClientServerUserGroupMembershipMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID, UInt16 groupOrTrackID, PositionSetStatusType status,
                const char *nicknameUTF8);
};

struct GPSClientServerRequestGroupDetailsMessage : GPSClientServerV2Message
{
    UInt16                          groupID;                        //  32: Group ID details are requested for
    MD5Type                         existingPNGDigest;              //  34: Allows server to update an existing PNG
```

```
                                                            //  50: Fixed size

    static GPSClientServerRequestGroupDetailsMessage
        *create (UInt32 sUDID, UUID128 certifiedUDID, UInt16 groupID, MD5Type existingPNGDigest);
};

struct GPSClientServerGroupDetailsMessage : GPSClientServerV2Message
{
    UInt16                          policy;                     //  32: User policy applied
    GroupDefinitionType             groupDefinition;            //  34: Same information like in groups list

    UInt32                          iconSize;                   //  44: Size of icon data starting data
    char                            data [0];                   //  48: Icon data followed by three zero terminated UTF8 encoded
strings
                                                            //  for group name, description, and owner
                                                            //  48: Variable size
};

struct GPSClientServerRequestUserDetailsMessage : GPSClientServerV2Message
{
    UInt16                          groupID;                    //  32: Group ID to decide if name or nickname returned for
    UUID128                         userUDID;                   //  34: User's UDID

    MD5Type                         existingPNGDigest;          //  50: Allows server to update an existing PNG
                                                            //  66: Fixed size
    static GPSClientServerRequestUserDetailsMessage *
        create (UInt32 sUDID, UUID128 certifiedUDID, UInt16 groupID, UUID128 userUDID, MD5Type existingPNGDigest);
};

struct GPSClientServerUserDetailsMessage : GPSClientServerV2Message
{
    UInt16                          groupID;                    //  32: Group ID
    UUID128                         userUDID;                   //  34: User's UDID

    UInt32                          iconSize;                   //  50: Size of icon data starting data
    char                            data [0];                   //  54: Icon data followed by two zero terminated UTF8 encoded
strings
                                                            //  for user name, status, and vehicle name
                                                            //  54: Variable size
};

struct GPSClientServerGroupListMessage : GPSClientServerV2Message
{
    UInt16                          numGroups;                  //  32: Number of list items below
    GroupDefinitionType             groups [0];                 //  34: List of groups available on server
                                                            //  34: Variable size
};

struct GPSClientServerRegisterGroupsAndTracksMessage : GPSClientServerV2Message
{
    UInt16                          groupAndTrackIDs [0];       //  32: NOTCERTIFIEDTRACKID terminated list
                                                            //  32: Variable size
};

#pragma pack(pop)

//  Protocols and callbacks
typedef void (*GPSMessageCallbackFctn) (GPSClientServerMessage *message, void *context);

//  Tracing support
const char *GPSMessageTypeName (GPSMessageType messageType);
const char *stringForMessage (GPSClientServerMessage *message, const char *actionName);

#endif
```