NAVILINK Protocol

This page is based on the Navilink protocol specification of Jun 30, 2005 provided by Locosys as PDF. Things found to be different in the real protocol implementation have been fixed in the specification below.

1. General Description

This document describes the NAVILINK Interface, which is used to communicate with a NAViGPS device. The NAVILINK Interface supports bi-directional transfer of data such as waypoints, routes, track data.

2. Physical Layer

The physical protocol is based on USB. The electrical characteristics are full duplex, serial data, 115200 baud, 8 data bits, no parity bits and 1 stop bit.

3. Packet Specification

Packet Format

The general packet of NAVILINK protocol is defined as follows:

Start Sequence	**Packet Length**	**Payload Packet**	**Checksum**	**End Sequence**
0xA0, 0xA2	Two-bytes (15-bits)	Up to 2^{15}-1	Two-bytes (15-bits)	0xB0, 0xB3

Bytes in a packet are transmitted low order byte first, followed by the high byte (little-endian order).

- Packet Length
  - The Packet length includes only payload bytes.
- Payload
  - The first byte of payload data is always the Packet ID (described in section 4).
- Checksum
  - The checksum is 15-bit checksum of the bytes in the payload data. The following pseudo code defines the algorithm used. Let `Packet` to be the array of bytes to be sent by the transport. Let `msgLen` be the number of bytes in the Packet array to be transmitted.

```
Index = first
checkSum = 0
while index < msgLen
    checkSum = checkSum + Packet[index]
checkSum = checkSum AND (0x7FFF)
```

4. Packet Summary

Below all packet types of the NaviLink protocol are described. For each type the sending direction is given (H→D, H←D or H↔D), where H is the Host computer and D the NAViGPS device.

For communicating with the device, the **NAVILINK Mode** needs to be enabled. In NAViGPS side, you can select function NAVILINK to set NAViGPS in link mode. You may exit from link mode by pressing any key or send a PID_QUITE packet to it. The NAViGPS will be reset after exiting from link mode.

Beginning and Ending Packets For a host to connecting to NAViGPS, the beginning packet PID_SYNC must be sent to establish the connection. Packet PID_ACK is received if the connection is OK. After the connection is established, the other packets then can be sent. Usually PID_QRY_CONFIG is sent to get the device configuration. Send packet PID_QUIT to end the connection.

Uploading Routes To upload a route, the waypoints referred in the route should be uploaded first. You may first back up all waypoints and routes then download new waypoints and new routes to guarantee data consistency.

PID_SYNC

The beginning packet to check if NAVi GPS device is ready or not.

- H→D
- Payload: 1 byte
- Byte 0: 0xd6
- Example: [A0 A2 01 00 d6 d6 00 B0 B3]

Expect to receive PID_ACK if NAViGPS is ready. The result can be failed if NAViGPS is not in link mode or USB connection is not ready.

PID_ACK

General acknowledge packet

- H↔D
- Payload: 1 byte
- Byte 0: 0x0c
- Example: [A0 A2 01 00 0c 0c 00 B0 B3]

PID_NAK

General none-acknowledge packet

- H↔D
- Payload: 1 byte
- Byte 0: 0x00
- Example: [A0 A2 01 00 00 00 00 B0 B3]

PID_QRY_INFORMATION

Packet to query NAViGPS information

- H→D
- Payload: 1 byte

- Byte 0: 0x20
- Example: [A0 A2 01 00 20 20 00 B0 B3]

Expect to receive a PID_DATA packet with payload data in T_INFORMATION type

PID_QRY_FW_VERSION

Packet to query the firmware version of the device

- H→D
- Payload: 1 byte
- Byte 0: 0xFE
- Example: [A0 A2 01 00 FE FE 00 B0 B3]

The device will return a PID_DATA packet holding the firmware version as a string.

PID_DATA

General data packet.

- H↔D
- Payload: up to 2^{15} bytes
- Byte 0: 0x03

PID_ADD_A_WAYPOINT

Packet to add a route to NAViGPS

- H→D
- Payload: 33
- Byte 0:0x3C
- Byte 1..32: waypoint data in T_WAYPOINT type

Expect to receive a PID_DATA packet with assigned waypoint ID if successful else PID_NAK is received

PID_QRY_WAYPOINTS

Packet to read 1 to 32 waypoints from NAVIGPS

- H→D
- Payload: 8 bytes
- Byte 0: 0x28
- Byte 1..4: the first waypoint to query by this packet, 0 based, waypoints are sorted by name
- Byte 5..6: number of waypoints to query (1..32)
- Byte 7: 0x01

Expect to receive PID_DATA packet with payload in T_WAYPOINTS, receive PID_NAK if no waypoints read.

For example, to read the first waypoint by sending: [A0 A2 08 00 28 00 00 00 00 01 00 01 2A 00 B0 B3]

To read the second and third waypoints by sending: [A0 A2 08 00 28 01 00 00 00 02 00 01 2C 00 B0 B3]

PID_QRY_ROUTE

Packet to query a route from NAViGPS

- H→D
- Payload: 8 bytes
- Byte 0: 0x24
- Byte 1..4: route number, 0..19, routes are sorted by name
- Byte 5..6: 0x0000
- Byte 7: always 0x1

Expect to receive PID_DATA with payload in T_ROUTE type

For example, to query the first route by sending: [A0 A2 08 00 24 00 00 00 00 00 00 01 25 00 B0 B3]

PID_DEL_WAYPOINT

Packet to delete one waypoint

- H→D
- Payload: 5 byte
- Byte 0: 0x36
- Byte 1..2: 0x0000
- Byte 3..4: waypoint ID(0..499)

Expect to receive PID_ACK if successful, else PID_NAK is received. The waypoint used by any routes cannot be deleted.

For example, to delete a waypoint with ID 01 by sending: [A0 A2 05 00 36 00 00 01 00 37 00 B0 B3]

PID_DEL_ALL_WAYPOINT

Packet request deleting all waypoints

- H→D
- Payload: 5 byte
- Byte 0: 0x37
- Byte 1..4: always 0x00f00000

Expect to receive PID_ACK if successful else PID_NAK is received. You have to delete all routes before deleting all waypoints.

For example, to delete all routes by sending: [A0 A2 05 00 37 00 00 f0 00 27 01 B0 B3]

PID_DEL_ROUTE

Packet to delete one route

- H→D

- Payload: 5 byte
- Byte 0: 0x34
- Byte 1..2: 0x0000
- Byte 3..4: route ID(0..19)

Expect to receive PID_ACK if successful, else PID_NAK is received.

For example, to delete a route with ID 01 by sending: [A0 A2 05 00 34 00 00 01 00 35 00 B0 B3]

PID_DEL_ALL_ROUTE

Packet request deleting all routes

- H→D
- Payload: 5 byte
- Byte 0: 0x35
- Byte 1..4: always 0x00f00000

Expect to receive PID_ACK if successful else PID_NAK is received.

For example, to delete all routes by sending [A0 A2 05 00 35 00 00 f0 00 25 01 B0 B3]

PID_ADD_A_ROUTE

Packet to add a route to NAViGPS

- H→D
- Payload : 1 + actual route length
- Byte 0: 0x3D
- Byte 1..n: route data in T_ROUTE type

Expect to receive PID_DATA with assigned route ID if successful else PID_NAK is received

PID_ERASE_TRACK

Packet request deleting track

- H→D
- Payload: 5 byte
- Byte 0: 0x11
- Byte 1..4: track buffer address, typical value is 0x400e0000, value can be found in T_INFORMATION.pTrackBuf
- Byte 5..6: 0x0000
- Byte 7 : always 0x00

Expect PID_CMD_OK in 3 seconds if successful.

Example: [A0 A2 08 00 11 00 00 0e 40 00 00 00 5F 00 B0 B3]

PID_READ_TRACKPOINTS

Packet request reading track logs from NAViGPS

- H→D
- Payload: 8 byte
- Byte0: 0x14
- Byte 1..4: start address (track buffer address+ offset)
- Byte 5..6: data length to read (32 ..512*32)
- Byte 7: always 0x00

Expect to receive PID_DATA with request track data in T_TRACKPOINT type in 4 seconds if successful. Send PID_ACK if data are received correctly.

For example, to read a track of 256[1)] points, 2 packets of PID_READ_TRACKPOINTS are needed. The start address and byte length are as follows:

Start address	Length
(0x400e0000+0)	32*256
(0x400e0000+32*256)	32*(256-2)

PID_WRITE_TRACKPOINTS

Packet request to write track points to NAVI

- H→D
- Payload: 8 byte
- Byte 0: 0x16
- Byte 1..4: start address(track buffer address + offset) typical value is 0x400e0000
- Byte 5..6: data length to write (32 ..127*32)
- Byte 7: always 0x00

After sending PID_WRITE_TRACKPOINTS packet, wait a short while (about 10ms seems to be enough) then send a PID_DATA packet with trackpoint data. Then you can expect to receive PID_CMD_OK within 4 seconds if successful. The maximum track points in one PID_DATA packet is 127. For example, to write a track of 520 points, 5 packets of PID_WRITE_TRACKPOINTS are needed.

PID_CMD_OK

Packet to indicate command is OK

- H←D
- Payload: 1 bytes
- Byte 0:0xf3
- Example:[A0 A2 01 00 f3 f3 00 B0 B3]

PID_CMD_FAIL

Packet to indicate command failed

- H←D
- Payload: 1 bytes

- Byte 0: 0xf4
- Example: [A0 A2 01 00 f4 f4 00 B0 B3]

PID_QUIT

Packet to end connection.

- H→D
- Payload: 1 bytes
- Byte 0: 0xf2
- Example: [A0 A2 01 00 f2 f2 00 B0 B3]

5. Data Type Definition

System Information

```
typedef struct {
    unsigned short totalWaypoint;        /* 0..1000 */
    unsigned char totalRoute;            /*0..20 */
    unsigned char totalTrack;            /* always 1 for NAViGPS */
    unsigned int startAdrOfTrackBuffer;
    unsigned int deviceSerialNum;
    unsigned short numOfTrackpoints;     /* 0..8191*/
    unsigned short protovolVersion;
    char unknown[16]; /* 16 bytes with unknown info */
    char username[16]; /* zero byte terminated */
} T_INFORMATION
```

Position

```
typedef struct {
    int latitude;              /*+-900000000,in 1/10000000 degree */
    int longitude;             /*+-1800000000,in 1/10000000 degree*/
    unsigned short altitude;   /*0..65535,in feet*/
} T_POSITION;
```

Size: 10 Bytes

Date & Time

```
typedef struct {
    unsigned char year;     /* actual year= year+2000 */
    unsigned char month;    /* 1..12 */
    unsigned char day;      /* 1..31 */
    unsigned char hour;     /* 0..23 */
    unsigned char minute;   /* 0..59 */
    unsigned char second;   /* 0..59 */
} T_DATETIME;
```

Size: 6 Bytes

Waypoint

```
typedef {
    unsigned short recordType; /* reserved. default 0x00 0x40*/
    unsigned short waypointID; /* 0..999*/
    char waypointName[7];      /* null-terminated- string,[ 0 .. 9 , , A .. Z ]*/
    unsigned char reserved;
    T_POSITION position;       /*position based on WGS84 datum*/
    T_DATETIME datetime;       /*time, date in UTC */
    unsigned char symbolType;  /*0..31*/
    unsigned char reserved;
    unsigned char tag1;        /*reserved, default 0x00 */
    unsigned char tag2;        /*reserved, default 0x7e */
} T_WAYPOINT
```

Size: 32 Bytes

Subroute

```
typedef {
    unsigned short recordType;     /*reserved, default 0x2010 */
    unsigned short waypointID[14]; /*0..999,0xffff:NULL waypoint ID */
    unsigned char tag1;            /*0x7f for last subroute*/
    unsigned char tag2;            /*reserved , default 0x77*/
} T_SUBROUTE
```

Route

```
typedef struct {
    unsigned short recordType; /*reserved, default 0x2000*/
    unsigned char routeID;     /*route ID:0..19,0xffff:null route ID*/
    unsigned char reserved;    /*default 0x20 */
    char routeName[14];        /*c string,[ 0 .. 9 , A .. Z , ]*/
    char reserved[2];
    unsigned int reserved;
    unsigned int reserved;
    unsigned short reserved;
    unsigned char flag;        /* reserved, default 0x7b */
    unsigned char mark;        /* reserved, default 0x77 */
    T_SUBROUTE subRoutes[9];
} T_ROUTE;
```

A route(in T_ROUTE type) consists a main route and 1 to maximum 9 subroutes. Its length is variable and depends on the number of subroutes included. The main route only describes the basic attributes. The waypoint IDs is kept in its subroutes. Each subroute (in T_SUBROUTE types) consists 1 to maximum 14 waypoint IDs. A null waypoint ID (0xffff) must be appended after the last waypoint ID(0..999) in the last subroute of a route.

If you happen to send a route with exactly 14 waypoints (or 28…) to the device, you may be better off adding another subroute, filled with 0xffff as waypoints. What is used by the NaviGPS in the second subroute added seems rather random to me. But the NaviGPS adds this second subroute too to represent such a route, so it may be considered good practice?

Track

```
typedef struct {
    unsigned short serialNum;      /*unique serial number,0..8191*/
    unsigned short headingOfPoint; /*0..360 degree*/
    int x;                         /*UTM x in WGS84 */
    int y;                         /*UTM y in WGS84 */
    T_POSITION position;           /*position in WGS84 datum*/
    T_DATETIME datetime;           /*time, date in UTC*/
    unsigned char zone;            /*UTM zone, 1..60*/
    unsigned char halfspeed;       /*in KMH, actual speed=halfspeed*2 */
    unsigned char tag1;            /*reserved, default 0x5a */
    unsigned char tag2;            /*reserved, default 0x7e */
} T_TRACKPOINT;
```

Please notice that track points in the track are limited to be in the same UTM zone.

Note: The UTM zone information seems not to be related to the trackpoint and always points to a place somewhere in China?

---

1) Fix Me! should this number be higher? 512 maybe?