NAVILINK Protocol

Jun 30, 2005

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

Hex	Packet ID	I/O*	Description
0xd6	PID_SYNC	H->D	The beginning packet to check if NAVi GPS device is ready or not. 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.
0x0c	PID_ACK	H<->D	General acknowledge packet Payload: 1 byte Byte 0:0x0c Example: [A0 A2 01 00 0c 0c 00 B0 B3]
0x00	PID_NAK	H<->D	General none-acknowledge packet Payload: 1 byte Byte 0:0x00 Example: [A0 A2 01 00 00 00 00 B0 B3]
0x20	PID_QRY_INFORMATION	H->D	Packet to query NAViGPS information 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
0x03	PID_DATA	H<->D	General data packet.
0x3C	PID_ADD_A_WAYPOINT	H->D	Packet to add a route to NAViGPS 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
0x28	PID_QRY_WAYPOINTS	H->D	Packet to read 1 to 32 waypoints from NAVIGPS Payload: 8 bytes Byte0:0x28 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) Bytes 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]
0X24	PID_QRY_ROUTE	H->D	Packet to query a route from NAViGPS Payload: 8 bytes Byte0:0x24 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]
0x36	PID_DEL_WAYPOINT	H->D	Packet to delete one waypoint Payload: 5 byte Byte0:0x36 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]
0x37	PID_DEL_ALL_WAYPOINT	H->D	Packet request deleting all waypoints Payload: 5 byte Byte0:0x37 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]
0x34	PID_DEL_ROUTE	H->D	Packet to delete one route Payload: 5 byte Byte0:0x34 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]
0x35	PID_DEL_ALL_ROUTE	H->D	Packet request deleting all routes Payload: 5 byte Byte0:0x35 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]
0x3D	PID_ADD_A_ROUTE	H->D	Packet to add a route to NAViGPS Payload : 1 + actual route length Byte 0:0x3D Byte 1..n:route data in TROUTE type Expect to receive PID_DATA with assigned route ID if successful else PID_NAK is received
0x11	PID_ERASE_TRACK	H->D	Packet request deleting track Payload: 5 byte Byte0:0x11 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]
0x14	PID_READ_TRACKPOINTS	H->D	Packet request reading track logs from NAViGPS 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 0x1 Expect to receivePID_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 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)
0x16	PID_WRITE_TRACKPOINTS	H->D	Packet request to write track points to NAVI Payload: 8 byte Byte0:0x14 Byte 1..4: start address(track buffer address+ offset) typical value is 0x400e0000 Byte 5..6: data length to wtite (32 ..127*32) Byte 7: always 0x01 After sending PID_WRITE_TRACKPOINTS packet, a PID_DATA packet with trackpoint data must be sent immediately. 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_WTITE_TRACKPOINTS are needed.
0xf3	PID_CMD_OK	H<-D	Packet to indicate command is OK Payload: 1 bytes Byte 0:0xf3 Example:[A0 A2 01 00 f3 f3 00 B0 B3]
0xf4	PID_CMD_FAIL	H<-D	Packet to indicate command failed Payload: 1 bytes Byte 0:0xf4 Example: [A0 A2 01 00 f4 f4 00 B0 B3]
0xf2	PID_QUIT	H->D	Packet to end connection. Payload: 1 bytes Byte 0:0xf2 Example: [A0 A2 01 00 f2 f2 00 B0 B3]

*H: Host D: NAViGPS device

- NAVILINK Mode

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.

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  username[16];
} 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;
```

- **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;
```

- **Waypoint**

```
typedef
{
unsigned short recordType;  /* reserved. default 0x4000*/
unsigned short waypointID;  /* 0..999*/
T_POSITION position;        /*position based on WGS84 datum*/
T_DATETIME datetime;        /*time, date in UTC */
unsigned char symbolType;   /*0..31*/
unsigned char  reserved;
char waypointName[7];       /* null-terminated- string,[‘0’..’9’,’ ‘,’A’..’Z’]*/
unsigned char reserved;
unsigned char tag1;         /*reserved, default 0x5a */
unsigned char tag2;         /*reserved, default 0x77 */
} T_WAYPOINT
```

- **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 char 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.

- **Track**

```
typedef  struct
{
  unsigned short  serialNum;     /*unique serial number,0..8191*/
  unsigned short headingOfPoint;/*0..360 degree*/
  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 */
   int x;                        /*UTM x in WGS84 */
   int y;                        /*UTM y in WGS84 */
  unsigned char tag1;            /*reserved, default 0x5a */
  unsigned char tag2;            /*reserved, default 0x77 */
  } T_TRACKPOINT;
```

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