Garmin Device Interface Specification

September 16, 2004
Drawing Number: 001-00063-00 Rev. B

Notice:

Garmin Corporation makes no warranties, express or implied, to companies or individuals accessing Garmin Corporation's Device Interface, or any other person, with respect to the Device Interface, including without limitation, any warranties of merchantability or fitness for a particular purpose, or arising from course of performance or trade usage, all of which are hereby excluded and disclaimed by Garmin Corporation.

Garmin Corporation shall not be liable for any indirect, incidental, consequential, punitive or special damages, even if Garmin Corporation has been advised of the possibility of such damages. Some states may not allow the exclusion on limitation of liability from consequential or incidental damages, so the foregoing limitation on liability for damages may not apply to you.

Warning:

All companies and individuals accessing the Device Interface are advised to ensure the correctness of their Device Interface software and to avoid the use of undocumented Device Interface features, particularly with respect to packet ID, command ID, and packet data content. Any software implementation errors or use of undocumented features, whether intentional or not, may result in damage to and/or unsafe operation of the device.

Technical Support Is Not Provided:

Garmin Corporation cannot provide technical support for questions relating to the Device Interface. However, if you would like to comment on this document, or if you would like to report a document error, you may send email to techsupp@garmin.com, or write to the address shown below.

Garmin Corporation
1200 E. 151st St.
Olathe, Kansas USA 66062
(913) 397-8200

Table of Contents

Table of Tables

1 Introduction

1.1 Overview

This document describes the Garmin Device Interface, which is used to communicate with a Garmin device. The Device Interface supports bi-directional transfer of data such as waypoints, routes, track logs, proximity waypoints, and satellite almanac. In the sections below, detailed descriptions of the interface protocols and data types are given, and differences among Garmin devices are identified.

1.2 Definition of Terms

In this document, “device” means a Garmin-produced device, and “host” means the device communicating with the Garmin-produced device. A host is usually a personal computer but is not required to be.

1.3 Specification of Data Types

All data types in this document are specified using the C programming language. Detailed specifications for basic C data types, basic Garmin data types, and device-specific data types are found in section 7 on page 18. Data types having limited scope are specified in earlier sections throughout this document (usually in the same section in which they are introduced). Unless otherwise specified, the behavior of software upon receiving invalid data is undefined.

2 Protocol Layers

The protocols used in the Garmin Device Interface are arranged in the following three layers:

Table 1 – Protocol Layers

Protocol Layer	
Application	(highest)
Link	
Physical	(lowest)

The Physical layer is based on RS-232. The Link layer uses packets with minimal overhead. At the Application layer, there are several protocols used to implement data transfers between a host and a device. These protocols are described in more detail later in this document.

3 Physical Protocols

3.1 Serial Protocol

The Serial Protocol is based on RS-232. The voltage characteristics are compatible with most hosts; however, the device transmits positive voltages only, whereas the RS-232 standard requires both positive and negative voltages. Also, the voltage swing between mark and space may not be large enough to meet the strict requirements of the RS-232 standard. Still, the device voltage characteristics are compatible with most hosts as long as the interface cable is wired correctly.

The other electrical characteristics are full duplex, serial data, 9600 baud, 8 data bits, no parity bits, and 1 stop bit.

The mechanical characteristics vary among devices; most devices have custom-designed interface connectors in order to meet Garmin packaging requirements. The electrical and mechanical connections to standard DB-9 or DB-25 connectors can be accomplished with special cables that are available from Garmin.

3.1.1 Serial Packet Format

All data is transferred in byte-oriented packets. A packet contains a three-byte header (DLE, ID, and Size), followed by a variable number of data bytes, followed by a three-byte trailer (Checksum, DLE, and ETX). The following table shows the format of a packet:

Table 2 – Serial Packet Format

Byte Number	Byte Description	Notes
0	Data Link Escape	ASCII DLE character (16 decimal)
1	Packet ID	identifies the type of packet
2	Size of Packet Data	number of bytes of packet data (bytes 3 to n-4)
3 to n-4	Packet Data	0 to 255 bytes
n-3	Checksum	2's complement of the sum of all bytes from byte 1 to byte n-4
n-2	Data Link Escape	ASCII DLE character (16 decimal)
n-1	End of Text	ASCII ETX character (3 decimal)

3.1.2 DLE Stuffing

If any byte in the Size, Packet Data, or Checksum fields is equal to DLE, then a second DLE is inserted immediately following the byte. This extra DLE is not included in the size or checksum calculation. This procedure allows the DLE character to be used to delimit the boundaries of a packet.

3.1.3 ACK/NAK Handshaking

Unless otherwise noted in this document, a device that receives a data packet must send an ACK or NAK packet to the transmitting device to indicate whether or not the data packet was successfully received. Normally, the transmitting device does not send any additional packets until an ACK or NAK is received (this is sometimes referred to as a "stop and wait" protocol).

The ACK packet has a Packet ID equal to 6 decimal (the ASCII ACK character), while the NAK packet has a Packet ID equal to 21 decimal (the ASCII NAK character). Both ACK and NAK packets contain an 8-bit integer in their packet data to indicate the Packet ID of the acknowledged packet. Note: some devices will report a Packet Data Size of two bytes for ACK and NAK packets; however, only the first byte should be considered. Note: Some devices may work sporadically if only one byte ACK/NAK packets are sent. The host should send two byte ACK/NAK packets to ensure consistency.

If an ACK packet is received, the data packet was received correctly and communication may continue. If a NAK packet is received, the data packet was not received correctly and should be sent again. NAKs are used only to indicate errors in the communications link, not errors in any higher-layer protocol. For example, consider the following higher-layer protocol error: a Pid_Wpt_Data packet was expected by the device, but a valid Pid_Xfer_Cmplt packet was received instead. This higher-layer protocol error does not cause the device to generate a NAK.

Some devices may send NAK packets during communication timeout conditions. For example, when the device is waiting for a packet in the middle of a protocol sequence, it will periodically send NAK packets (typically every 2-5 seconds) if no data is received from the host. The purpose of this NAK Packet is to guard against a deadlock condition in which the host is waiting for an ACK or NAK in response to a data packet that was never received by the device (perhaps due to cable disconnection during the middle of a protocol sequence). Not all devices provide NAKs during timeout conditions, so the host should not rely on this behavior. It is recommended that the host implement its own timeout and retransmission strategy to guard against deadlock. For example, if the host does not receive an ACK within a reasonable amount of time, it could warn the user and give the option of aborting or re-initiating the transfer.

3.1.4 Serial Protocol Packet IDs

The Serial Protocol Packet ID values are defined using the enumerations shown below:

```
enum
   {
   Pid_Ack_Byte            =   6,
   Pid_Nak_Byte            =  21
   };
```

Additional Packet IDs are defined by other Link protocols (see below); however, the values of ASCII DLE (16 decimal) and ASCII ETX (3 decimal) are reserved and will never be used as Packet IDs in any Link protocol. This allows more efficient detection of packet boundaries in the link-layer software implementation.

3.2 USB Protocol

This protocol provides a mechanism for using the link and application layer protocols over USB.

3.2.1 USB Protocol Details

Microsoft Windows application developers do not need to be familiar with the concepts in this section in order to use the USB protocol.

The host always transmits to the device over the Bulk OUT pipe.

The device can choose to transmit to the host over either the Interrupt IN pipe or the Bulk IN pipe. Once the device begins an application protocol over a particular pipe, the device will complete the protocol over that same pipe. Some devices may transmit data to the host only using the Interrupt IN pipe.

The host must constantly check the interrupt pipe for data. The host only reads the bulk pipe when it receives a Data Available packet from the device (see section 3.2.3.1 below). Once the host begins reading the bulk pipe, it should keep reading packets until it receives a zero length transfer (i.e. USB transfer, not a Garmin packet.)

3.2.2 USB Packet Format

All packets transferred using this protocol have the following format:

Table 3 – USB Packet Format

Byte Number	Byte Description	Notes
0	Packet Type	USB Protocol Layer = 0, Application Layer = 20
1-3	Reserved	Must be set to 0
4-5	Packet ID	
6-7	Reserved	Must be set to 0
8-11	Data Size	
12+	Data	

3.2.3 USB Protocol Layer Packet Ids

The USB Protocol Packet ID values are defined using the enumerations shown below:

```
enum
   {
   Pid_Data_Available      = 2,
   Pid_Start_Session       = 5,
   Pid_Session_Started     = 6
   };
```

3.2.3.1 Data Available Packet

The Data Available packet signifies that data has become available for the host to read. The host should read data until receiving a transfer with no data (zero length). No data is associated with this packet.

Table 4 – Data Available Packet

N	Direction	Packet ID	Packet Data Type
0	Device to Host	Pid_Data_Available	n/a

3.2.3.2 Start Session Packet

The Start Session packet must be sent by the host to begin transferring packets over USB. It must also be sent anytime the host deliberately stops transferring packets continuously over USB and wishes to begin again. No data is associated with this packet.

Table 5 – Start Session Packet

N	Direction	Packet ID	Packet Data Type
0	Host to Device	Pid_Start_Session	n/a

3.2.3.3 Session Started Packet

The Session Started packet indicates that transfers can take place to and from the device. The host should ignore any packets it receives before receiving this packet. The data returned with this packet is the device's unit ID.

Table 6 – Session Started Packet

N	Direction	Packet ID	Packet Data Type
0	Device to Host	Pid_Session_Started	uint32

3.2.4 Garmin USB Driver for Microsoft Windows

This section provides information related to the use of the Garmin-provided USB driver for use on Microsoft Windows operating systems. This driver is compatible with Windows 98, ME, 2000 and XP. It is assumed that the reader is familiar with programming for the Windows Platform Software Development Kit and Driver Development Kit.

Applications send packets to the device using the Win32 WriteFile function. If the packet size is an exact multiple of the USB packet size, an additional call to WriteFile should be made passing in no data.

Applications receive packets asynchronously from the device by constantly calling the Win32 DeviceIoControl function. When an application receives a Data Available packet, it should read packets using the Win32 ReadFile function. Once an application begins receiving packets for a protocol using DeviceIoControl or ReadFile, all subsequent packets for that protocol will be received using the same function.

3.2.4.1 Device Interface GUID

```
// {2C9C45C2-8E7D-4C08-A12D-816BBAE722C0}
DEFINE_GUID(GUID_DEVINTERFACE_GRMNUSB, 0x2c9c45c2L, 0x8e7d, 0x4c08, 0xa1, 0x2d, 0x81,
0x6b, 0xba, 0xe7, 0x22, 0xc0);
```

3.2.4.2 Constants

```
#define API_VERSION        1
#define MAX_BUFFER_SIZE    4096
#define ASYNC_DATA_SIZE    64
```

3.2.4.3 ReadFile, WriteFile Functions

The buffer passed in by the client to ReadFile or WriteFile must be no larger than `MAX_BUFFER_SIZE`. If data exceeds `MAX_BUFFER_SIZE`, multiple calls must be made.

3.2.4.4 IOCTLS

The following constants are intended for use with the DeviceIoControl function. For each IOCTL below, the return value is the number of bytes written to the output buffer.

```
#define IOCTL_API_VERSION CTL_CODE( FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED,
FILE_ANY_ACCESS )
```

Output buffer receives 4-byte API version.

```
#define IOCTL_ASYNC_IN CTL_CODE( FILE_DEVICE_UNKNOWN, 0x850, METHOD_BUFFERED,
FILE_ANY_ACCESS )
```

Output buffer receives asynchronous data from the device. Size is equal to or less than `ASYNC_DATA_SIZE`. The client should constantly have a call into the driver with this IOCTL. The driver stores a limited amount of asynchronous data.

```
#define IOCTL_USB_PACKET_SIZE CTL_CODE( FILE_DEVICE_UNKNOWN, 0x851, METHOD_BUFFERED,
FILE_ANY_ACCESS )
```

Output buffer receives 4-byte USB packet size. Client is responsible for sending a zero length transfer if the amount of data sent to the device is an integral multiple of the USB packet size.

4 Link Protocols

4.1 L000 – Basic Link Protocol

All devices implement the Basic Link Protocol. Its primary purpose is to facilitate initial communication between the host and the device using the Product Data Protocol (see section 6.1 on page 8), which allows the host to determine which type of device is connected. Using this knowledge, the host can then determine which device-specific Link protocol to use for all other communication with the device.

4.1.1 Basic Packet IDs

The Basic Packet ID values are defined using the enumerations shown below:

```
enum
  {
  Pid_Protocol_Array      = 253,        /* may not be implemented in all devices */
  Pid_Product_Rqst        = 254,
  Pid_Product_Data        = 255,
  Pid_Ext_Product_Data    = 248         /* may not be implemented in all devices */
  };
```

4.2 L001 – Link Protocol 1

This Link protocol is used for the majority of devices (see section 8.2 on page 45). This protocol is the same as L000 – Basic Link Protocol, except that the following Packet IDs are used in addition to the Basic Packet IDs:

```
enum
  {
  Pid_Command_Data        =  10,
  Pid_Xfer_Cmplt          =  12,
  Pid_Date_Time_Data      =  14,
  Pid_Position_Data       =  17,
  Pid_Prx_Wpt_Data        =  19,
  Pid_Records             =  27,
  Pid_Rte_Hdr             =  29,
  Pid_Rte_Wpt_Data        =  30,
  Pid_Almanac_Data        =  31,
  Pid_Trk_Data            =  34,
  Pid_Wpt_Data            =  35,
  Pid_Pvt_Data            =  51,
  Pid_Rte_Link_Data       =  98,
  Pid_Trk_Hdr             =  99,
  Pid_FlightBook_Record   = 134,         /* packet with FlightBook data  */
  Pid_Lap                 = 149,         /* part of Forerunner data */
  Pid_Wpt_Cat             = 152
  };
```

4.3 L002 – Link Protocol 2

This Link protocol is used mainly for panel-mounted aviation devices (see section 8.2 on page 45). This protocol is the same as L000 – Basic Link Protocol, except that the following Packet IDs are used in addition to the Basic Packet IDs:

```
enum
  {
  Pid_Almanac_Data        =  4,
  Pid_Command_Data        = 11,
  Pid_Xfer_Cmplt          = 12,
  Pid_Date_Time_Data      = 20,
  Pid_Position_Data       = 24,
  Pid_Prx_Wpt_Data        = 27,
  Pid_Records             = 35,
  Pid_Rte_Hdr             = 37,
  Pid_Rte_Wpt_Data        = 39,
  Pid_Wpt_Data            = 43
  };
```

5 Overview of Application Protocols

Each Application protocol has a unique Protocol ID to allow it to be identified apart from the others. Future devices may introduce additional protocols to transfer new data types or to provide a newer version of an existing protocol (e.g., protocol A101 might be introduced as a newer version of protocol A100). Whenever a new protocol is introduced, it is expected that the host software will have to be updated to accommodate the new protocol. However, new devices may continue to support some of the older protocols, so full or partial communication may still be possible with older host software. To better support this capability, newer devices are able to report which protocols they support (see section 6.2 on page 9). In all other cases, the host must contain a lookup table to determine which protocols to use with which device types (see section 8.2 on page 45).

5.1 Undocumented Application Packets

A device may transmit application packets containing packet IDs that are not documented in this specification. These packets are used for internal testing purposes by Garmin engineering. Their contents are subject to change at any time and should not be used by third-party applications for any purpose. They should be handled according to the physical protocols described in this specification and then discarded.

5.2 Packet Sequences

Each of the Application protocols is defined in terms of a packet sequence, which defines the order and types of packets exchanged between two devices, including direction of the packet, Packet ID, and packet data type. An example of a packet sequence is shown below:

Table 7 – Example Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_First	First_Data_Type
1	Device1 to Device2	Pid_Second	ignored
2	Device1 to Device2	Pid_Third	<D0>
3	Device2 to Device1	Pid_Fourth	<D1>
4	Device2 to Device1	Pid_Fifth	<D2>

In this example, there are five packets exchanged: three from Device1 to Device2 and two in the other direction. Each of these five packets must be acknowledged, but the acknowledgement packets are omitted from the table for clarity. Most of the protocols are symmetric, meaning that the protocol for transfers in one direction (e.g., Device to Host) is the same as the protocol for transfers in the other direction (e.g., Host to Device). For symmetric protocols, either the device or the host may assume the role of Device1 or Device2. For non-symmetric protocols, the sequence table will explicitly show the roles of the device and host instead of showing Device1 and Device2.

The first column of the table shows the packet number (used only for reference; this number is not encoded into the packet). The second column shows the direction of each packet transfer. The third column shows the Packet ID enumeration name (to determine the actual value for a Packet ID, see section 3.2.3 on page 3). The last column shows the Packet Data Type.

5.3 Packet Data Types

The Packet Data Type may be specified in several different ways. First, it may be specified with an explicitly-named data type (e.g., "First_Data_Type"); all explicitly-named data types are defined in this document. Second, it may indicate that the packet data is not used (e.g., "ignored"), in which case the packet data may have a zero size. Finally, the data type for a packet may be specified using angle-bracket notation (e.g. <D0>). This notation indicates that the data type is device-specific. In the example above, there are three device-specific data types (<D0>, <D1>, and <D2>).

These device-specific data types must be determined dynamically by the host depending on which type of device is currently connected. For older devices, this determination is made through the use of a lookup table within the host (see section 8.2 on page 45), however, newer devices are able to dynamically report their protocols and data types (see section 6.2 on page 9).

5.4 Standard Beginning and Ending Packets

Many Application protocols use standard beginning and ending packets called Pid_Records and Pid_Xfer_Cmplt, respectively, as shown in the table below:

Table 8 – Standard Beginning and Ending Packets

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
…	…	…	…
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first packet (Packet 0) provides Device2 with an indication of the number of data packets to follow, excluding the Pid_Xfer_Cmplt packet (i.e., Packet 1 through n-2). This allows Device2 to monitor the progress of the transfer. The last packet (Packet n-1) indicates that the transfer is complete. This last packet also contains data to indicate which kind of transfer has been completed in the Command_Id_Type data type (see section 6.3 on page 10).

The Command_Id_Type value for each kind of transfer matches the command ID used to initiate that kind of transfer (see section 6.3 on page 10). As a result, the actual Command_Id_Type value depends on which Device Command protocol is implemented by the device. Because of this dependency, enumeration names (not values) for Command_Id_Type are given in the description of each Application protocol later in this document.

5.4.1 Records_Type

The Records_Type contains a 16-bit integer that indicates the number of data packets to follow, excluding the Pid_Xfer_Cmplt packet. The type definition for the Records_Type is shown below:

```
typedef uint16 Records_Type;
```

5.5 Device Overwriting of Identically-Named Data

When receiving data from the host, most devices will erase identically-named data and replace it with the new data received from the host. For example, if the host sends a waypoint named XYZ, most devices will overwrite the waypoint named XYZ that was previously stored in device memory. No warning is sent from the device prior to overwriting identically-named data.

Some devices (e.g., the StreetPilot) have special handling for identically-named waypoints. These devices compare the position of the incoming waypoint with the position of the existing waypoint (Note: altitude is ignored during the comparison). If the positions match, the device will erase the identically-named waypoint and replace it with the new waypoint received from the host. If the positions differ, the device will create a new, unique name for the incoming waypoint and preserve the existing waypoint under the original name. There is no mechanism available for the host to determine which method a device uses for waypoints (overwriting vs. unique naming).

6 Application Protocols

6.1 A000 – Product Data Protocol

All devices are required to implement the Product Data Protocol using the default physical and basic link protocols described earlier in this document. The Product Data Protocol is used to query the device to find out its Product ID, which is then used by the host to determine which data transfer protocols are supported by the connected device (see section 8.2 on page 45).

The packet sequence for the Product Data Protocol is shown below:

Table 9 – A000 Protocol Data Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Host to Device	Pid_Product_Rqst	ignored
1	Device to Host	Pid_Product_Data	Product_Data_Type
2	Device to Host	Pid_Ext_Product_Data	Ext_Product_Data_Type
…	…	…	…
N-1	Device to Host	Pid_Ext_Product_Data	Ext_Product_Data_Type

Packet 0 (Pid_Product_Rqst) is a special product request packet that is sent to the device. Packet 1 (Pid_Product_Data) is returned to the host and contains data to identify the device, which is provided in the data type Product_Data_Type. Packets 2 (Pid_Ext_Product_Data) through N-1 (Pid_Ext_Product_Data) are not implemented by all devices and contain additional information about the device as provided in the data type Ext_Product_Data_Type.

6.1.1 Product_Data_Type

The Product_Data_Type contains two 16-bit integers followed by one or more null-terminated strings. The first integer indicates the Product ID, and the second integer indicates the software version number multiplied by 100 (e.g., version 3.11 will be indicated by 311 decimal). Following these integers, there will be one or more null-terminated strings. The first string provides a textual description of the device and its software version; this string is intended to be displayed by the host to the user in an “about” dialog box. The host should ignore all subsequent strings; they are used during manufacturing to identify other properties of the device and are not formatted for display to the end user.

The type definition for the Product_Data_Type is shown below:

```
typedef struct
   {
   uint16                  product_ID;
   sint16                  software_version;
/* char                    product_description[];  null-terminated string */
/* ...                     zero or more additional null-terminated strings */
   } Product_Data_Type;
```

6.1.2 Ext_Product_Data_Type

The Ext_Product_Data_Type contains zero or more null-terminated strings. The host should ignore all these strings; they are used during manufacturing to identify other properties of the device and are not formatted for display to the end user.

```
typedef struct
   {
/* ...                     zero or more additional null-terminated strings */
   } Ext_Product_Data_Type;
```

6.2 A001 – Protocol Capability Protocol

The Protocol Capability Protocol is a one-way protocol that allows a device to report its protocol capabilities and device-specific data types to the host. When this protocol is supported by the device, it is automatically initiated by the device immediately after completion of the Product Data Protocol. Using this protocol, the host obtains a list of all protocols and data types supported by the device.

The packet sequence for the Protocol Capability Protocol is shown below:

Table 10 – A001 Protocol Capability Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device to Host	Pid_Protocol_Array	Protocol_Array_Type

Packet 0 (Pid_Protocol_Array) contains an array of Protocol_Data_Type structures, each of which contains tag-encoded protocol information.

The order of array elements is used to associate data types with protocols. For example, a protocol that requires two data types <D0> and <D1> is indicated by a tag-encoded protocol ID followed by two tag-encoded data type IDs, where the first data type ID identifies <D0> and the second data type ID identifies <D1>.

6.2.1 Protocol_Array_Type

The Protocol_Array_Type is an array of Protocol_Data_Type structures. The number of Protocol_Data_Type structures contained in the array is determined by observing the size of the received packet data.

```
typedef Protocol_Data_Type Protocol_Array_Type[];
```

6.2.2 Protocol_Data_Type

The Protocol_Data_Type is comprised of a one-byte tag field and a two-byte data field. The tag identifies which kind of ID is contained in the data field, and the data field contains the actual ID.

```
typedef struct
   {
   uint8                   tag;
   uint16                  data;
   } Protocol_Data_Type;
```

The combination of tag value and data value must correspond to one of the protocols or data types specified in this document. For example, this document specifies a Waypoint Transfer Protocol identified as "A100." This protocol is represented by a tag value of 'A' and a data field value of 100.

6.2.3 Tag Values for Protocol_Data_Type

The enumerated values for the tag member of the Protocol_Data_Type are shown below. The characters shown are translated to numeric values using the ASCII character set.

```
enum
  {
  Tag_Phys_Prot_Id        = 'P',        /* tag for Physical protocol ID */
  Tag_Link_Prot_Id        = 'L',        /* tag for Link protocol ID */
  Tag_Appl_Prot_Id        = 'A',        /* tag for Application protocol ID */
  Tag_Data_Type_Id        = 'D'         /* tag for Data Type ID */
  };
```

6.2.4 Protocol Capabilities Example

The following table shows a series of three-byte records that might be received by a host during the Protocol Capabilities Protocol:

Table 11 – Protocol Capabilities Example

Tag (byte 0)	Data (bytes 1 & 2)	Notes
'L'	1	Device supports Link Protocol 1 (L001)
'A'	10	Device supports Device Command Protocol 1 (A010)
'A'	100	Device supports the Waypoint Transfer Protocol (A100)
'D'	100	Device uses Data Type D100 for <D0> during waypoint transfer
'A'	200	Device supports the Route Transfer Protocol (A200)
'D'	200	Device uses Data Type D200 for <D0> during route transfer
'D'	100	Device uses Data Type D100 for <D1> during route transfer
'A'	300	Device supports the Track Log Transfer Protocol (A300)
'D'	300	Device uses Data Type D300 for <D0> during track log transfer
'A'	500	Device supports the Almanac Transfer Protocol (A500)
'D'	500	Device uses Data Type D500 for <D0> during almanac transfer

The device omits the following protocols from the above transmission:

A000 – Product Data Protocol
A001 – Protocol Capability Protocol

A000 is omitted because all devices support it. A001 is omitted because it is the very protocol being used to communicate the protocol information.

6.3 Device Command Protocols

This section describes a group of similar protocols known as Device Command protocols. These protocols are used to send commands to a device; for example, the host might command the device to transmit its waypoints. All devices are required to implement one of the Device Command protocols, although some commands may not be implemented by the device (reception of an unimplemented command causes no error in the device; it simply ignores the command). The only difference among Device Command protocols is that the enumerated values for the Command_Id_Type are different (see the section for each Device Command protocol below).

Note that either the host or device is allowed to initiate a transfer without a command from the other device (for example, when the host transfers data to the device, or when the user presses buttons on the device to initiate a transfer).

The packet sequence for each Device Command protocol is shown below:

Table 12 – Device Command Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Command_Data	Command_Id_Type

Packet 0 (Pid_Command_Data) contains data to indicate a command, which is provided in the data type Command_Id_Type. The Command_Id_Type contains a 16-bit integer that indicates a particular command. The type definition for Command_Id_Type is shown below:

```
typedef uint16 Command_Id_Type;
```

6.3.1 A010 – Device Command Protocol 1

This protocol is implemented by the majority of devices (see section 8.2 on page 45). The enumerated values for Command_Id_Type are shown below:

```
enum
   {
   Cmnd_Abort_Transfer     =  0,        /* abort current transfer */
   Cmnd_Transfer_Alm       =  1,        /* transfer almanac */
   Cmnd_Transfer_Posn      =  2,        /* transfer position */
   Cmnd_Transfer_Prx       =  3,        /* transfer proximity waypoints */
   Cmnd_Transfer_Rte       =  4,        /* transfer routes */
   Cmnd_Transfer_Time      =  5,        /* transfer time */
   Cmnd_Transfer_Trk       =  6,        /* transfer track log */
   Cmnd_Transfer_Wpt       =  7,        /* transfer waypoints */
   Cmnd_Turn_Off_Pwr       =  8,        /* turn off power */
   Cmnd_Start_Pvt_Data     = 49,        /* start transmitting PVT data */
   Cmnd_Stop_Pvt_Data      = 50,        /* stop transmitting PVT data */
   Cmnd_FlightBook_Transfer= 92,        /* transfer flight records */
   Cmnd_Transfer_Laps      = 117,       /* transfer laps */
   Cmnd_Transfer_Wpt_Cats  = 121        /* transfer waypoint categories */
   };
```

Note: The "Cmnd_Turn_Off_Pwr" command may not be acknowledged by the device.

Note: The PC can send Cmnd_Abort_Transfer in the middle of a transfer of data to the device in order to cancel the transfer.

6.3.2 A011 – Device Command Protocol 2

This protocol is implemented mainly by panel-mounted aviation devices (see section 8.2 on page 45). The enumerated values for Command_Id_Type are shown below:

```
enum
   {
   Cmnd_Abort_Transfer     =  0,        /* abort current transfer */
   Cmnd_Transfer_Alm       =  4,        /* transfer almanac */
   Cmnd_Transfer_Rte       =  8,        /* transfer routes */
   Cmnd_Transfer_Prx       = 17,        /* transfer proximity waypoints */
   Cmnd_Transfer_Time      = 20,        /* transfer time */
   Cmnd_Transfer_Wpt       = 21,        /* transfer waypoints */
   Cmnd_Turn_Off_Pwr       = 26         /* turn off power */
   };
```

6.4 A100 – Waypoint Transfer Protocol

The Waypoint Transfer Protocol is used to transfer waypoints between devices. When the host commands the device to send waypoints, the device will send every waypoint stored in its database. When the host sends waypoints to the device, the host may selectively transfer any waypoint it chooses.

The packet sequence for the Waypoint Transfer Protocol is shown below:

Table 13 – A100 Waypoint Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Wpt_Data	<D0>
2	Device1 to Device2	Pid_Wpt_Data	<D0>
…	…	…	…
n-2	Device1 to Device2	Pid_Wpt_Data	<D0>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Wpt, which is also the command value used by the host to initiate a transfer of waypoints from the device.

Packets 1 through n-2 (Pid_Wpt_Data) each contain data for one waypoint, which is provided in device-specific data type <D0>. This data type usually contains an identifier string, latitude and longitude, and other device-specific data.

6.5 A101 – Waypoint Category Transfer Protocol

The Waypoint Category Transfer Protocol is used to transfer waypoint categories between devices. When a device is commanded to send waypoint categories, the device will send every waypoint category stored in its database.

The packet sequence for the Waypoint Category Transfer Protocol is shown below:

Table 14 – A101 Waypoint Category Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Wpt_Cat	<D0>
2	Device1 to Device2	Pid_Wpt_Cat	<D0>
…	…	…	…
n-2	Device1 to Device2	Pid_Wpt_Cat	<D0>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Wpt_Cats, which is also the command value used by the host to initiate a transfer of waypoint categories from the device.

Packets 1 through n-2 (Pid_Wpt_Cat) each contain data for one waypoint category, which is provided in device-specific data type <D0>. The order of packets 1 through n-2 indicates the association of the data received with a particular category. For example, packet 1 contains data associated with category 1, packet 3 is associated with category 3, etc. Each device will be capable of containing some maximum number of waypoint categories. If a device receives more data packets than its maximum then it should ignore those data packets beyond its maximum.

6.6 Route Transfer Protocol

The Route Transfer Protocol is used to transfer routes between devices. When the host commands the device to send routes, the device will send every route stored in its database. When the host sends routes to the device, the host may selectively transfer any route it chooses.

6.6.1 Database Matching for Route Waypoints

Certain devices contain an internal database of waypoint information; for example, most aviation devices have an internal database of aviation waypoints, and the StreetPilot has an internal database of land waypoints. When routes are being transferred from the host to one of these devices, the device will attempt to match the incoming route waypoints with waypoints in its internal database. First, the device inspects the “wpt_class” member of the incoming route waypoint; if it indicates a non-user waypoint, then the device searches its internal database using values contained in other members of the route waypoint. For aviation devices, the “ident” and “cc” members are used to search the internal database; for the StreetPilot, the “subclass” member is used to search the internal database. If a match is found, the waypoint from the internal database is used for the route; otherwise, a new user waypoint is created and used for the route.

6.6.2 A200 – Route Transfer Protocol

The packet sequence for the A200 Route Transfer Protocol is shown below:

Table 15 – A200 Route Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Rte_Hdr	<D0>
2	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
3	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
…	…	…	…
n-2	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Rte, which is also the command value used by the host to initiate a transfer of routes from the device.

Packet 1 (Pid_Rte_Hdr) contains route header information, which is provided in device-specific data type <D0>. This data type usually contains information that uniquely identifies the route. Packets 2 through n-2 (Pid_Rte_Wpt_Data) each contain data for one route waypoint, which is provided in device-specific data type <D1>. This data type usually contains the same waypoint data that is transferred in the Waypoint Transfer Protocol.

More than one route can be transferred during the protocol by sending another set of packets that resemble Packets 1 through n-2 in the table above. This additional set of packets is sent immediately after the previous set of route packets. In other words, it is not necessary to send Pid_Xfer_Cmplt until all route packets have been sent for the multiple routes. Device2 must monitor the Packet ID to detect the beginning of a new route, which is indicated by a Packet ID equal to Pid_Rte_Hdr. Any number of routes may be transferred in this fashion.

6.6.3 A201 – Route Transfer Protocol

The packet sequence for the A201 Route Transfer Protocol is shown below:

Table 16 – A201 Route Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Rte_Hdr	<D0>
2	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
3	Device1 to Device2	Pid_Rte_Link_Data	<D2>
4	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
5	Device1 to Device2	Pid_Rte_Link_Data	<D2>
…	…	…	…
n-2	Device1 to Device2	Pid_Rte_Wpt_Data	<D1>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Rte, which is also the command value used by the host to initiate a transfer of routes from the device.

Packet 1 (Pid_Rte_Hdr) contains route header information, which is provided in device-specific data type <D0>. This data type usually contains information that uniquely identifies the route. Even numbered packets starting with packet 2 contain data for one route waypoint, which is provided in device-specific data type <D1>. Odd numbered packets starting with packet 3 and excluding packet n-1 (Pid_Xfer_Cmplt) contain data for one link between the adjacent waypoints. This link data is provided in device-specific data type <D2>.

More than one route can be transferred during the protocol by sending another set of packets that resemble Packets 1 through n-2 in the table above. This additional set of packets is sent immediately after the previous set of route packets. In other words, it is not necessary to send Pid_Xfer_Cmplt until all route packets have been sent for the multiple routes. Device2 must monitor the Packet ID to detect the beginning of a new route, which is indicated by a Packet ID equal to Pid_Rte_Hdr. Any number of routes may be transferred in this fashion.

6.7 Track Log Transfer Protocol

6.7.1 Time Values Ignored by Device

When the host transfers a track log to the device, the device ignores the incoming time value for each track log point and sets the time value to zero in its internal database. If the device later transfers the track log back to the host, the time values will be zero. Thus, the host is able to differentiate between track logs that were actually recorded by the device and track logs that were transferred to the device by an external host.

NOTE: Some devices use 0x7FFFFFFF or 0xFFFFFFFF instead of zero to indicate an invalid time value.

6.7.2 A300 – Track Log Transfer Protocol

The Track Log Transfer Protocol is used to transfer track logs between devices. Most devices store only one track log (called the "active" track log), however, some newer devices can store multiple track logs (in addition to the active track log). When the host commands the device to send track logs, the device will concatenate all track logs (i.e., the active track log plus any stored track logs) to form one track log consisting of multiple segments; i.e., the protocol does not provide a way for the host to request selective track logs from the device, nor is there a way for the host to decompose the concatenated track log into its original set of track logs. When the host sends track logs to the device, the track log is always stored in the active track log within the device; i.e., there is no way to transfer track logs into the database of stored track logs. None of these limitations affect devices that store only one track log.

The packet sequence for the Track Log Transfer Protocol is shown below:

Table 17 – A300 Track Log Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Trk_Data	<D0>
2	Device1 to Device2	Pid_Trk_Data	<D0>
…	…	…	…
n-2	Device1 to Device2	Pid_Trk_Data	<D0>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Trk, which is also the command value used by the host to initiate a transfer of track logs from the device.

Packets 1 through n-2 (Pid_Trk_Data) each contain data for one track log point, which is provided in device-specific data type <D0>. This data type usually contains four elements: latitude, longitude, time, and a boolean flag indicating whether the point marks the beginning of a new track log segment.

6.7.3 A301 – Track Log Transfer Protocol

The packet sequence for the Track Log Transfer Protocol is shown below:

Table 18 – A301 Track Log Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Trk_Hdr	<D0>
2	Device1 to Device2	Pid_Trk_Data	<D1>
3	Device1 to Device2	Pid_Trk_Data	<D1>
…	…	…	…
n-2	Device1 to Device2	Pid_Trk_Data	<D1>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Trk, which is also the command value used by the host to initiate a transfer of track logs from the device.

Packet 1 (Pid_Trk_Hdr) contains track header information, which is provided in device-specific data type <D0>. This data type usually contains information that uniquely identifies the track log. Packets 2 through n-2 (Pid_Trk_Data) each contain data for one track log point, which is provided in device-specific data type <D1>.

More than one track log can be transferred during the protocol by sending another set of packets that resemble packets 1 through n-2 in the table above. This additional set of packets is sent immediately after the previous set of track log packets. In other words, it is not necessary to send Pid_Xfer_Cmplt until all track log packets have been sent for the multiple track logs. Device2 must monitor the Packet ID to detect the beginning of a new track log, which is indicated by a Packet ID of Pid_Trk_Hdr. Any number of track logs may be transferred in this fashion.

6.7.4 A302 – Track Log Transfer Protocol

Identical to A301, but used in the Forerunner line of devices.

6.8 A400 – Proximity Waypoint Transfer Protocol

The Proximity Waypoint Transfer Protocol is used to transfer proximity waypoints between devices. When the host commands the device to send proximity waypoints, the device will send all proximity waypoints stored in its database. When the host sends proximity waypoints to the device, the host may selectively transfer any proximity waypoint it chooses.

The packet sequence for the Proximity Waypoint Transfer Protocol is shown below:

Table 19 – A400 Proximity Waypoint Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Prx_Wpt_Data	<D0>
2	Device1 to Device2	Pid_Prx_Wpt_Data	<D0>
…	…	…	…
n-2	Device1 to Device2	Pid_Prx_Wpt_Data	<D0>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Prx, which is also the command value used by the host to initiate a transfer of proximity waypoints from the device.

Packets 1 through n-2 (Pid_Prx_Wpt_Data) each contain data for one proximity waypoint, which is provided in device-specific data type <D0>. This data type usually contains the same waypoint data that is transferred during the Waypoint Transfer Protocol, plus a valid proximity alarm distance.

Some devices (e.g. aviation panel mounts) require a delay of one or more seconds between proximity waypoints when the host transfers proximity waypoints to the device.

6.9 A500 – Almanac Transfer Protocol

The Almanac Transfer Protocol is used to transfer almanacs between devices. The main purpose of this protocol is to allow a host to update a device that has been in storage for more than six months, or has undergone a memory clear operation. To avoid a potentially lengthy auto-initialization sequence, the device must have current almanac, approximate date and time, and approximate position. Thus, after transferring an almanac to the device, the host should subsequently transfer the date, time, and position (in that order) to the device using the following protocols: A600 – Date and Time Initialization Protocol (see section 6.10 on page 16), and A700 – Position Initialization Protocol (see section 6.12 on page 17). After receiving the almanac, the device may transmit a request for time and/or a request for position using one of the Device Command protocols.

The device is also able to transmit almanac to the host, allowing the user to archive the almanac or transfer the almanac to another device.

The packet sequence for the Almanac Transfer Protocol is shown below:

Table 20 – A500 Almanac Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Records	Records_Type
1	Device1 to Device2	Pid_Almanac_Data	<D0>
2	Device1 to Device2	Pid_Almanac_Data	<D0>
…	…	…	…
n-2	Device1 to Device2	Pid_Almanac_Data	<D0>
n-1	Device1 to Device2	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Alm, which is also the command value used by the host to initiate a transfer of the almanac from the device

Packets 1 through n-2 (Pid_Almanac_Data) each contain almanac data for one satellite, which is provided in device-specific data type <D0>. This data type contains data that describes the satellite's orbit characteristics.

Some device-specific data types (<D0>) do not include a satellite ID to relate each data packet to a particular satellite in the GPS constellation. For these data types, Device1 must transmit exactly 32 Pid_Almanac_Data packets, and these packets must be sent in PRN order (i.e., the first packet contains data for PRN-01 and so on up to PRN-32). If the data for a particular satellite is missing or if the satellite is non-existent, then the week number for that satellite must be set to a negative number to indicate that the data is invalid.

6.10 A600 – Date and Time Initialization Protocol

The Date and Time Initialization Protocol is used to transfer the current date and time between devices. This is normally done in conjunction with transferring an almanac to the device (see section 6.9 on page 16).

The packet sequence for the Date and Time Initialization Protocol is shown below:

Table 21 – A600 Date and Time Initialization Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Date_Time_Data	<D0>

Packet 0 (Pid_Date_Time_Data) contains date and time data, which is provided in device-specific data type <D0>.

6.11 A650 – FlightBook Transfer Protocol

The FlightBook Transfer Protocol is used to transfer auto-generated FlightBook data to the host.

The packet sequence for the FlightBook Transfer Protocol is shown below:

Table 22 – A650 FlightBook Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Host to Device	Pid_Command_Data	Command_Id_Type
1	Device to Host	Pid_Records	Records_Type
2	Device to Host	Pid_FlightBook_Record	<D0>
…	…	…	...
n-2	Device to Host	Pid_FlightBook_Record	<D0>
n-1	Device to Host	Pid_Xfer_Cmplt	Command_Id_Type

Packet 0 (Pid_Command_Data) commands the device to initiate a FlightBook transfer. Packets 1 and n-1 are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value in packets 0 and n-1 is Cmnd_FlightBook_Transfer. Packets 2 through n-2 each contain a FlightBook record using device-specific data type <D0>.

6.12 A700 – Position Initialization Protocol

The Position Initialization Protocol is used to transfer the current position between devices. This is normally done in conjunction with transferring an almanac to the device (see section 6.9 on page 16).

The packet sequence for the Position Initialization Protocol is shown below:

Table 23 – A700 Position Initialization Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device1 to Device2	Pid_Position_Data	<D0>

Packet 0 (Pid_Position_Data) contains position data, which is provided in device-specific data type <D0>. The device may ignore the position data provided by this protocol whenever the device has a valid position fix or whenever the device is in simulator mode.

6.13 A800 – PVT Protocol

The PVT Protocol is used to provide the host with real-time position, velocity, and time (PVT), which is transmitted by the device approximately once per second. This protocol is provided as an alternative to NMEA so that the user may permanently choose the Garmin format on the device instead of switching back and forth between NMEA format and Garmin format.

The host can turn PVT on or off by using a Device Command Protocol (see section 6.3 on page 10). PVT is turned on when the host sends the Cmnd_Start_Pvt_Data command and is turned off when the host sends the Cmnd_Stop_Pvt_Data command. Note that, as a side effect, most devices turn off PVT whenever they respond to the Product Data Protocol.

ACK and NAK packets are optional for this protocol; however, unlike other protocols, the device will not retransmit a PVT packet in response to receiving a NAK from the host.

The packet sequence for the PVT Protocol is shown below:

Table 24 – A800 PVT Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device to Host (ACK/NAK optional)	Pid_Pvt_Data	<D0>

Packet 0 (Pid_Pvt_Data) contains position, velocity, and time data, which is provided in device-specific data type <D0>.

6.14 A906 – Lap Transfer Protocol

The packet sequence for the Lap Transfer Protocol is shown below:

Table 25 – A906 Lap Transfer Protocol Packet Sequence

N	Direction	Packet ID	Packet Data Type
0	Device to Host	Pid_Records	Records_Type
1	Device to Host	Pid_Lap	<D0>
2	Device to Host	Pid_Lap	<D0>
…	…	…	...
n-2	Device to Host	Pid_Lap	<D0>
n-1	Device to Host	Pid_Xfer_Cmplt	Command_Id_Type

The first and last packets (Packet 0 and Packet n-1) are the standard beginning and ending packets (see section 5.4 on page 7). The Command_Id_Type value contained in Packet n-1 is Cmnd_Transfer_Laps, which is also the command value used by the host to initiate a transfer of laps from the device.

Packets 1 through n-2 (Pid_Lap) each contain data for one lap, which is provided in device-specific data type <D0>.

7 Data Types

7.1 Serialization of Data

Every data type must be serialized into a stream of bytes for transferal over a serial data link. Serialization of each data type is accomplished by transmitting the bytes in the order that they would occur in memory given a machine with the following characteristics: 1) data structure members are stored in memory in the same order as they appear in the type definition; 2) all structures are packed, meaning that there are no unused “pad” bytes between structure members; 3) multibyte numeric types are stored in memory using little-endian format, meaning the least-significant byte occurs first in memory followed by increasingly significant bytes in successive memory locations.

7.2 Character Sets

Unless otherwise noted, all devices use characters from the ASCII character set. Each string type is limited to a specific subset of ASCII characters as shown below:

Table 26 – Character Sets

User Waypoint Identifier:	upper-case letters, numbers
Waypoint Comment:	upper-case letters, numbers, space, hyphen
Route Comment:	upper-case letters, numbers, space, hyphen
City:	ignored by device
State:	ignored by device
Facility Name:	ignored by device
Country Code:	upper-case letters, numbers, space
Route Identifier:	upper-case letters, numbers, space, hyphen
Route Waypoint Identifier:	any ASCII character
Link Identifier:	any ASCII character
Track Identifier:	upper-case letters, numbers, space, hyphen

Some devices may allow additional characters beyond those mentioned above, but no attempt is made in this document to identify these device-specific additions. The host should be prepared to receive any ASCII character from the device, but only transmit the characters shown above back to the device.

7.3 Basic Data Types

The following are basic data types that are used in the definition of more complex data types.

7.3.1 char

The char data type is 8 bits in size and its value is an ASCII character.

7.3.2 Character Arrays

Unless otherwise noted, all character arrays are padded with spaces and are not required to have a null terminator. For example, consider the following data type:

```
char  xyz[6];  /* xyz type */
```

The word "CAT" would be stored in this data type as shown below:

```
xyz[0] = 'C';
xyz[1] = 'A';
xyz[2] = 'T';
xyz[3] = ' ';
xyz[4] = ' ';
xyz[5] = ' ';
```

Character arrays provide a way to transfer strings between the host and the device. However, the size of a character array may exceed the number of characters that a device has allotted for the string being transferred. If this is the case, the device will ignore any characters beyond the size of its allotted string. For example, a "cmnt" character array may allow 40 characters to be transferred, but a device may only have 16 characters allotted for a "cmnt" string. In this case, the device will ignore the last 24 characters of the transferred character array.

7.3.3 Variable-Length Strings

In contrast to character arrays, a variable-length string is a null-terminated string that can be any length as long it does not cause a data packet to become larger than the maximum allowable data packet size. When a variable-length string is a member of a data structure, the data type is specified as follows:

```
typedef struct
   {
   sint16                    abc;
/* char                      xyz[]           null-terminated string */
   sint16                    def;
   } example_type;
```

This syntax indicates that a variable-length string named xyz occurs between the abc and def members of the data structure. Therefore, the address offset (from the beginning of the data structure) of the def member cannot be known until run-time (after the variable-length string is decoded). Whenever possible, variable-length strings are placed at the end of a data structure to minimize the need for run-time address offset calculations.

7.3.4 uint8

The uint8 data type is used for 8-bit unsigned integers.

7.3.5 uint16

The uint16 data type is used for 16-bit unsigned integers.

7.3.6 uint32

The uint32 data type is used for 32-bit unsigned integers.

7.3.7 sint16

The sint16 data type is used for 16-bit signed integers.

7.3.8 sint32

The sint32 data type is used for 32-bit signed integers.

7.3.9 float32

The float32 data type is 32-bit IEEE-format floating point data (1 sign bit, 8 exponent bits, and 23 mantissa bits).

7.3.10 float64

The float64 data type is 64-bit IEEE-format floating point data (1 sign bit, 11 exponent bits, and 52 mantissa bits).

7.3.11 bool

The bool data type is an 8-bit integer used to indicate true (non-zero) or false (zero).

7.3.12 position_type

The position_type is used to indicate latitude and longitude in semicircles, where 2^{31} semicircles equal 180 degrees. North latitudes and East longitudes are indicated with positive numbers; South latitudes and West longitudes are indicated with negative numbers.

```
typedef struct
   {
   sint32                    lat;         /* latitude in semicircles */
   sint32                    lon;         /* longitude in semicircles */
   } position_type;
```

The following formulas show how to convert between degrees and semicircles:

degrees = semicircles * (180 / 2^{31})
semicircles = degrees * (2^{31} / 180)

7.3.13 radian_position_type

The radian_position_type is used to indicate latitude and longitude in radians, where π radians equal 180 degrees. North latitudes and East longitudes are indicated with positive numbers; South latitudes and West longitudes are indicated with negative numbers.

```
typedef struct
   {
   float64                  lat;          /* latitude in radians */
   float64                  lon;          /* longitude in radians */
   } radian_position_type;
```

The following formulas show how to convert between degrees and radians:

degrees = radians * (180 / π)
radians = degrees * (π / 180)

7.3.14 time_type

The time_type is used in some data structures to indicate an absolute time. It is an unsigned 32 bit integer and its value is the number of seconds since 12:00 am December 31, 1989 UTC.

7.3.15 symbol_type

The symbol_type is used in certain devices to indicate the symbol for a waypoint:

```
typedef uint16 symbol_type;
```

The enumerated values for symbol_type are shown below. Note that most devices that use this type are limited to a much smaller subset of these symbols, and no attempt is made in this document to indicate which subsets are valid for each of these devices. However, the device will ignore any disallowed symbol values that are received and instead substitute the value for a generic dot symbol. Therefore, there is no harm in attempting to use any value shown in the table below except that the device may not accept the requested value.

```
enum
    {
    /*------------------------------------------------------------
    Marine symbols
    ------------------------------------------------------------*/
    sym_anchor          =   0,  /* white anchor symbol              */
    sym_bell            =   1,  /* white bell symbol                */
    sym_diamond_grn     =   2,  /* green diamond symbol             */
    sym_diamond_red     =   3,  /* red diamond symbol               */
    sym_dive1           =   4,  /* diver down flag 1                */
    sym_dive2           =   5,  /* diver down flag 2                */
    sym_dollar          =   6,  /* white dollar symbol              */
    sym_fish            =   7,  /* white fish symbol                */
    sym_fuel            =   8,  /* white fuel symbol                */
    sym_horn            =   9,  /* white horn symbol                */
    sym_house           =  10,  /* white house symbol               */
    sym_knife           =  11,  /* white knife & fork symbol        */
    sym_light           =  12,  /* white light symbol               */
    sym_mug             =  13,  /* white mug symbol                 */
    sym_skull           =  14,  /* white skull and crossbones symbol*/
    sym_square_grn      =  15,  /* green square symbol              */
    sym_square_red      =  16,  /* red square symbol                */
    sym_wbuoy           =  17,  /* white buoy waypoint symbol       */
    sym_wpt_dot         =  18,  /* waypoint dot                     */
    sym_wreck           =  19,  /* white wreck symbol               */
    sym_null            =  20,  /* null symbol (transparent)        */
    sym_mob             =  21,  /* man overboard symbol             */
    sym_buoy_ambr       =  22,  /* amber map buoy symbol            */
    sym_buoy_blck       =  23,  /* black map buoy symbol            */
    sym_buoy_blue       =  24,  /* blue map buoy symbol             */
    sym_buoy_grn        =  25,  /* green map buoy symbol            */
    sym_buoy_grn_red    =  26,  /* green/red map buoy symbol        */
    sym_buoy_grn_wht    =  27,  /* green/white map buoy symbol      */
    sym_buoy_orng       =  28,  /* orange map buoy symbol           */
    sym_buoy_red        =  29,  /* red map buoy symbol              */
    sym_buoy_red_grn    =  30,  /* red/green map buoy symbol        */
    sym_buoy_red_wht    =  31,  /* red/white map buoy symbol        */
    sym_buoy_violet     =  32,  /* violet map buoy symbol           */
    sym_buoy_wht        =  33,  /* white map buoy symbol            */
    sym_buoy_wht_grn    =  34,  /* white/green map buoy symbol      */
    sym_buoy_wht_red    =  35,  /* white/red map buoy symbol        */
    sym_dot             =  36,  /* white dot symbol                 */
    sym_rbcn            =  37,  /* radio beacon symbol              */
    sym_boat_ramp       = 150,  /* boat ramp symbol                 */
    sym_camp            = 151,  /* campground symbol                */
    sym_restrooms       = 152,  /* restrooms symbol                 */
    sym_showers         = 153,  /* shower symbol                    */
    sym_drinking_wtr    = 154,  /* drinking water symbol            */
    sym_phone           = 155,  /* telephone symbol                 */
    sym_1st_aid         = 156,  /* first aid symbol                 */
    sym_info            = 157,  /* information symbol               */
    sym_parking         = 158,  /* parking symbol                   */
    sym_park            = 159,  /* park symbol                      */
    sym_picnic          = 160,  /* picnic symbol                    */
    sym_scenic          = 161,  /* scenic area symbol               */
    sym_skiing          = 162,  /* skiing symbol                    */
    sym_swimming        = 163,  /* swimming symbol                  */
    sym_dam             = 164,  /* dam symbol                       */
    sym_controlled      = 165,  /* controlled area symbol           */
    sym_danger          = 166,  /* danger symbol                    */
    sym_restricted      = 167,  /* restricted area symbol           */
    sym_null_2          = 168,  /* null symbol                      */
    sym_ball            = 169,  /* ball symbol                      */
```

```
sym_car             = 170,  /* car symbol                        */
sym_deer            = 171,  /* deer symbol                       */
sym_shpng_cart      = 172,  /* shopping cart symbol              */
sym_lodging         = 173,  /* lodging symbol                    */
sym_mine            = 174,  /* mine symbol                       */
sym_trail_head      = 175,  /* trail head symbol                 */
sym_truck_stop      = 176,  /* truck stop symbol                 */
sym_user_exit       = 177,  /* user exit symbol                  */
sym_flag            = 178,  /* flag symbol                       */
sym_circle_x        = 179,  /* circle with x in the center       */
sym_open_24hr       = 180,  /* open 24 hours symbol              */
sym_fhs_facility    = 181,  /* U Fishing Hot Spots™ Facility     */
sym_bot_cond        = 182,  /* Bottom Conditions                 */
sym_tide_pred_stn   = 183,  /* Tide/Current Prediction Station   */
sym_anchor_prohib   = 184,  /* U anchor prohibited symbol        */
sym_beacon          = 185,  /* U beacon symbol                   */
sym_coast_guard     = 186,  /* U coast guard symbol              */
sym_reef            = 187,  /* U reef symbol                     */
sym_weedbed         = 188,  /* U weedbed symbol                  */
sym_dropoff         = 189,  /* U dropoff symbol                  */
sym_dock            = 190,  /* U dock symbol                     */
sym_marina          = 191,  /* U marina symbol                   */
sym_bait_tackle     = 192,  /* U bait and tackle symbol          */
sym_stump           = 193,  /* U stump symbol                    */
/*-----------------------------------------------------------------
User customizable symbols
The values from sym_begin_custom to sym_end_custom inclusive are
reserved for the identification of user customizable symbols.
-----------------------------------------------------------------*/
sym_begin_custom   = 7680,  /* first user customizable symbol    */
sym_end_custom     = 8191,  /* last user customizable symbol     */
/*-----------------------------------------------------------------
Land symbols
-----------------------------------------------------------------*/
sym_is_hwy         = 8192,  /* interstate hwy symbol             */
sym_us_hwy         = 8193,  /* us hwy symbol                     */
sym_st_hwy         = 8194,  /* state hwy symbol                  */
sym_mi_mrkr        = 8195,  /* mile marker symbol                */
sym_trcbck         = 8196,  /* TracBack (feet) symbol            */
sym_golf           = 8197,  /* golf symbol                       */
sym_sml_cty        = 8198,  /* small city symbol                 */
sym_med_cty        = 8199,  /* medium city symbol                */
sym_lrg_cty        = 8200,  /* large city symbol                 */
sym_freeway        = 8201,  /* intl freeway hwy symbol           */
sym_ntl_hwy        = 8202,  /* intl national hwy symbol          */
sym_cap_cty        = 8203,  /* capitol city symbol (star)        */
sym_amuse_pk       = 8204,  /* amusement park symbol             */
sym_bowling        = 8205,  /* bowling symbol                    */
sym_car_rental     = 8206,  /* car rental symbol                 */
sym_car_repair     = 8207,  /* car repair symbol                 */
sym_fastfood       = 8208,  /* fast food symbol                  */
sym_fitness        = 8209,  /* fitness symbol                    */
sym_movie          = 8210,  /* movie symbol                      */
sym_museum         = 8211,  /* museum symbol                     */
sym_pharmacy       = 8212,  /* pharmacy symbol                   */
sym_pizza          = 8213,  /* pizza symbol                      */
sym_post_ofc       = 8214,  /* post office symbol                */
sym_rv_park        = 8215,  /* RV park symbol                    */
sym_school         = 8216,  /* school symbol                     */
sym_stadium        = 8217,  /* stadium symbol                    */
sym_store          = 8218,  /* dept. store symbol                */
sym_zoo            = 8219,  /* zoo symbol                        */
sym_gas_plus       = 8220,  /* convenience store symbol          */
```

```
sym_faces          = 8221,  /* live theater symbol              */
sym_ramp_int       = 8222,  /* ramp intersection symbol         */
sym_st_int         = 8223,  /* street intersection symbol       */
sym_weigh_sttn     = 8226,  /* inspection/weigh station symbol  */
sym_toll_booth     = 8227,  /* toll booth symbol                */
sym_elev_pt        = 8228,  /* elevation point symbol           */
sym_ex_no_srvc     = 8229,  /* exit without services symbol     */
sym_geo_place_mm   = 8230,  /* Geographic place name, man-made  */
sym_geo_place_wtr  = 8231,  /* Geographic place name, water     */
sym_geo_place_lnd  = 8232,  /* Geographic place name, land      */
sym_bridge         = 8233,  /* bridge symbol                    */
sym_building       = 8234,  /* building symbol                  */
sym_cemetery       = 8235,  /* cemetery symbol                  */
sym_church         = 8236,  /* church symbol                    */
sym_civil          = 8237,  /* civil location symbol            */
sym_crossing       = 8238,  /* crossing symbol                  */
sym_hist_town      = 8239,  /* historical town symbol           */
sym_levee          = 8240,  /* levee symbol                     */
sym_military       = 8241,  /* military location symbol         */
sym_oil_field      = 8242,  /* oil field symbol                 */
sym_tunnel         = 8243,  /* tunnel symbol                    */
sym_beach          = 8244,  /* beach symbol                     */
sym_forest         = 8245,  /* forest symbol                    */
sym_summit         = 8246,  /* summit symbol                    */
sym_lrg_ramp_int   = 8247,  /* large ramp intersection symbol   */
sym_lrg_ex_no_srvc = 8248,  /* large exit without services smbl */
sym_badge          = 8249,  /* police/official badge symbol     */
sym_cards          = 8250,  /* gambling/casino symbol           */
sym_snowski        = 8251,  /* snow skiing symbol               */
sym_iceskate       = 8252,  /* ice skating symbol               */
sym_wrecker        = 8253,  /* tow truck (wrecker) symbol       */
sym_border         = 8254,  /* border crossing (port of entry)  */
sym_geocache       = 8255,  /* geocache location                */
sym_geocache_fnd   = 8256,  /* found geocache                   */
sym_cntct_smiley   = 8257,  /* Rino contact symbol, "smiley"    */
sym_cntct_ball_cap = 8258,  /* Rino contact symbol, "ball cap"  */
sym_cntct_big_ears = 8259,  /* Rino contact symbol, "big ear"   */
sym_cntct_spike    = 8260,  /* Rino contact symbol, "spike"     */
sym_cntct_goatee   = 8261,  /* Rino contact symbol, "goatee"    */
sym_cntct_afro     = 8262,  /* Rino contact symbol, "afro"      */
sym_cntct_dreads   = 8263,  /* Rino contact symbol, "dreads"    */
sym_cntct_female1  = 8264,  /* Rino contact symbol, "female 1"  */
sym_cntct_female2  = 8265,  /* Rino contact symbol, "female 2"  */
sym_cntct_female3  = 8266,  /* Rino contact symbol, "female 3"  */
sym_cntct_ranger   = 8267,  /* Rino contact symbol, "ranger"    */
sym_cntct_kung_fu  = 8268,  /* Rino contact symbol, "kung fu"   */
sym_cntct_sumo     = 8269,  /* Rino contact symbol, "sumo"      */
sym_cntct_pirate   = 8270,  /* Rino contact symbol, "pirate"    */
sym_cntct_biker    = 8271,  /* Rino contact symbol, "biker"     */
sym_cntct_alien    = 8272,  /* Rino contact symbol, "alien"     */
sym_cntct_bug      = 8273,  /* Rino contact symbol, "bug"       */
sym_cntct_cat      = 8274,  /* Rino contact symbol, "cat"       */
sym_cntct_dog      = 8275,  /* Rino contact symbol, "dog"       */
sym_cntct_pig      = 8276,  /* Rino contact symbol, "pig"       */
sym_hydrant        = 8282,  /* water hydrant symbol             */
sym_flag_blue      = 8284,  /* blue flag symbol                 */
sym_flag_green     = 8285,  /* green flag symbol                */
sym_flag_red       = 8286,  /* red flag symbol                  */
sym_pin_blue       = 8287,  /* blue pin symbol                  */
sym_pin_green      = 8288,  /* green pin symbol                 */
sym_pin_red        = 8289,  /* red pin symbol                   */
sym_block_blue     = 8290,  /* blue block symbol                */
sym_block_green    = 8291,  /* green block symbol               */
```

```
sym_block_red      = 8292,  /* red block symbol                */
sym_bike_trail     = 8293,  /* bike trail symbol               */
sym_circle_red     = 8294,  /* red circle symbol               */
sym_circle_green   = 8295,  /* green circle symbol             */
sym_circle_blue    = 8296,  /* blue circle symbol              */
sym_diamond_blue   = 8299,  /* blue diamond symbol             */
sym_oval_red       = 8300,  /* red oval symbol                 */
sym_oval_green     = 8301,  /* green oval symbol               */
sym_oval_blue      = 8302,  /* blue oval symbol                */
sym_rect_red       = 8303,  /* red rectangle symbol            */
sym_rect_green     = 8304,  /* green rectangle symbol          */
sym_rect_blue      = 8305,  /* blue rectangle symbol           */
sym_square_blue    = 8308,  /* blue square symbol              */
sym_letter_a_red   = 8309,  /* red letter 'A' symbol           */
sym_letter_b_red   = 8310,  /* red letter 'B' symbol           */
sym_letter_c_red   = 8311,  /* red letter 'C' symbol           */
sym_letter_d_red   = 8312,  /* red letter 'D' symbol           */
sym_letter_a_green = 8313,  /* green letter 'A' symbol         */
sym_letter_c_green = 8314,  /* green letter 'C' symbol         */
sym_letter_b_green = 8315,  /* green letter 'B' symbol         */
sym_letter_d_green = 8316,  /* green letter 'D' symbol         */
sym_letter_a_blue  = 8317,  /* blue letter 'A' symbol          */
sym_letter_b_blue  = 8318,  /* blue letter 'B' symbol          */
sym_letter_c_blue  = 8319,  /* blue letter 'C' symbol          */
sym_letter_d_blue  = 8320,  /* blue letter 'D' symbol          */
sym_number_0_red   = 8321,  /* red number '0' symbol           */
sym_number_1_red   = 8322,  /* red number '1' symbol           */
sym_number_2_red   = 8323,  /* red number '2' symbol           */
sym_number_3_red   = 8324,  /* red number '3' symbol           */
sym_number_4_red   = 8325,  /* red number '4' symbol           */
sym_number_5_red   = 8326,  /* red number '5' symbol           */
sym_number_6_red   = 8327,  /* red number '6' symbol           */
sym_number_7_red   = 8328,  /* red number '7' symbol           */
sym_number_8_red   = 8329,  /* red number '8' symbol           */
sym_number_9_red   = 8330,  /* red number '9' symbol           */
sym_number_0_green = 8331,  /* green number '0' symbol         */
sym_number_1_green = 8332,  /* green number '1' symbol         */
sym_number_2_green = 8333,  /* green number '2' symbol         */
sym_number_3_green = 8334,  /* green number '3' symbol         */
sym_number_4_green = 8335,  /* green number '4' symbol         */
sym_number_5_green = 8336,  /* green number '5' symbol         */
sym_number_6_green = 8337,  /* green number '6' symbol         */
sym_number_7_green = 8338,  /* green number '7' symbol         */
sym_number_8_green = 8339,  /* green number '8' symbol         */
sym_number_9_green = 8340,  /* green number '9' symbol         */
sym_number_0_blue  = 8341,  /* blue number '0' symbol          */
sym_number_1_blue  = 8342,  /* blue number '1' symbol          */
sym_number_2_blue  = 8343,  /* blue number '2' symbol          */
sym_number_3_blue  = 8344,  /* blue number '3' symbol          */
sym_number_4_blue  = 8345,  /* blue number '4' symbol          */
sym_number_5_blue  = 8346,  /* blue number '5' symbol          */
sym_number_6_blue  = 8347,  /* blue number '6' symbol          */
sym_number_7_blue  = 8348,  /* blue number '7' symbol          */
sym_number_8_blue  = 8349,  /* blue number '8' symbol          */
sym_number_9_blue  = 8350,  /* blue number '9' symbol          */
sym_triangle_blue  = 8351,  /* blue triangle symbol            */
sym_triangle_green = 8352,  /* green triangle symbol           */
sym_triangle_red   = 8353,  /* red triangle symbol             */
/*-----------------------------------------------------------------
Aviation symbols
-----------------------------------------------------------------*/
sym_airport        = 16384, /* airport symbol                  */
sym_int            = 16385, /* intersection symbol             */
```

```
   sym_ndb            = 16386, /* non-directional beacon symbol    */
   sym_vor            = 16387, /* VHF omni-range symbol            */
   sym_heliport       = 16388, /* heliport symbol                  */
   sym_private        = 16389, /* private field symbol             */
   sym_soft_fld       = 16390, /* soft field symbol                */
   sym_tall_tower     = 16391, /* tall tower symbol                */
   sym_short_tower    = 16392, /* short tower symbol               */
   sym_glider         = 16393, /* glider symbol                    */
   sym_ultralight     = 16394, /* ultralight symbol                */
   sym_parachute      = 16395, /* parachute symbol                 */
   sym_vortac         = 16396, /* VOR/TACAN symbol                 */
   sym_vordme         = 16397, /* VOR-DME symbol                   */
   sym_faf            = 16398, /* first approach fix               */
   sym_lom            = 16399, /* localizer outer marker           */
   sym_map            = 16400, /* missed approach point            */
   sym_tacan          = 16401, /* TACAN symbol                     */
   sym_seaplane       = 16402, /* Seaplane Base                    */
   };
```

7.4 Product-Specific Data Types

Note that all positions are referenced to WGS-84. All altitudes are referenced to the WGS-84 geoid.

7.4.1 D100_Wpt_Type

```
typedef struct
  {
  char                    ident[6];    /* identifier */
  position_type           posn;        /* position */
  uint32                  unused;      /* should be set to zero */
  char                    cmnt[40];    /* comment */
  } D100_Wpt_Type;
```

7.4.2 D101_Wpt_Type

```
typedef struct
  {
  char                    ident[6];    /* identifier */
  position_type           posn;        /* position */
  uint32                  unused;      /* should be set to zero */
  char                    cmnt[40];    /* comment */
  float32                 dst;         /* proximity distance (meters) */
  uint8                   smbl;        /* symbol id */
  } D101_Wpt_Type;
```

The enumerated values for the “smbl” member of the D101_Wpt_Type are the same as those for symbol_type (see section 7.3.15 on page 21). However, since the “smbl” member of the D101_Wpt_Type is only 8-bits (instead of 16-bits), all symbol_type values whose upper byte is non-zero are disallowed in the D101_Wpt_Type.

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.3 D102_Wpt_Type

```
typedef struct
  {
  char                    ident[6];    /* identifier */
  position_type           posn;        /* position */
  uint32                  unused;      /* should be set to zero */
  char                    cmnt[40];    /* comment */
  float32                 dst;         /* proximity distance (meters) */
  symbol_type             smbl;        /* symbol id */
  } D102_Wpt_Type;
```

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.4 D103_Wpt_Type

```
typedef struct
   {
   char                     ident[6];     /* identifier */
   position_type            posn;         /* position */
   uint32                   unused;       /* should be set to zero */
   char                     cmnt[40];     /* comment */
   uint8                    smbl;         /* symbol id */
   uint8                    dspl;         /* display option */
   } D103_Wpt_Type;
```

The enumerated values for the “smbl” member of the D103_Wpt_Type are shown below:

```
enum
   {
   smbl_dot                 =  0,         /* dot symbol */
   smbl_house               =  1,         /* house symbol */
   smbl_gas                 =  2,         /* gas symbol */
   smbl_car                 =  3,         /* car symbol */
   smbl_fish                =  4,         /* fish symbol */
   smbl_boat                =  5,         /* boat symbol */
   smbl_anchor              =  6,         /* anchor symbol */
   smbl_wreck               =  7,         /* wreck symbol */
   smbl_exit                =  8,         /* exit symbol */
   smbl_skull               =  9,         /* skull symbol */
   smbl_flag                = 10,         /* flag symbol */
   smbl_camp                = 11,         /* camp symbol */
   smbl_circle_x            = 12,         /* circle with x symbol */
   smbl_deer                = 13,         /* deer symbol */
   smbl_1st_aid             = 14,         /* first aid symbol */
   smbl_back_track          = 15          /* back track symbol */
   };
```

The enumerated values for the “dspl” member of the D103_Wpt_Type are shown below:

```
enum
   {
   dspl_name                = 0,          /* Display symbol with waypoint name */
   dspl_none                = 1,          /* Display symbol by itself */
   dspl_cmnt                = 2           /* Display symbol with comment */
   };
```

7.4.5 D104_Wpt_Type

```
typedef struct
   {
   char                     ident[6];     /* identifier */
   position_type            posn;         /* position */
   uint32                   unused;       /* should be set to zero */
   char                     cmnt[40];     /* comment */
   float32                  dst;          /* proximity distance (meters) */
   symbol_type              smbl;         /* symbol id */
   uint8                    dspl;         /* display option */
   } D104_Wpt_Type;
```

The enumerated values for the “dspl” member of the D104_Wpt_Type are shown below:

```
enum
   {
   dspl_smbl_none          = 0,         /* Display symbol by itself  */
   dspl_smbl_only          = 1,         /* Display symbol by itself  */
   dspl_smbl_name          = 3,         /* Display symbol with waypoint name */
   dspl_smbl_cmnt          = 5,         /* Display symbol with comment  */
   };
```

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.6 D105_Wpt_Type

```
typedef struct
   {
   position_type           posn;        /* position */
   symbol_type             smbl;        /* symbol id */
/* char                    wpt_ident[];    null-terminated string */
   } D105_Wpt_Type;
```

7.4.7 D106_Wpt_Type

```
typedef struct
   {
   uint8                   wpt_class;   /* class */
   uint8                   subclass[13];/* subclass */
   position_type           posn;        /* position */
   symbol_type             smbl;        /* symbol id */
/* char                    wpt_ident[];    null-terminated string */
/* char                    lnk_ident[];    null-terminated string */
   } D106_Wpt_Type;
```

The enumerated values for the “wpt_class” member of the D106_Wpt_Type are as follows:

Zero:	indicates a user waypoint (“subclass” is ignored).
Non-zero:	indicates a non-user waypoint (“subclass” must be valid).

For non-user waypoints (such as a city in the device map database), the device will provide a non-zero value in the “wpt_class” member, and the “subclass” member will contain valid data to further identify the non-user waypoint. If the host wishes to transfer this waypoint back to the device (as part of a route), the host must leave the “wpt_class” and “subclass” members unmodified. For user waypoints, the host must ensure that the “wpt_class” member is zero, but the “subclass” member will be ignored and should be set to zero.

The “lnk_ident” member provides a string that indicates the name of the path from the previous waypoint in the route to this one. For example, “HIGHWAY 101” might be placed in “lnk_ident” to show that the path from the previous waypoint to this waypoint is along Highway 101. The “lnk_ident” string may be empty (i.e., no characters other than the null terminator), which indicates that no particular path is specified.

7.4.8 D107_Wpt_Type

```
typedef struct
   {
   char                    ident[6];    /* identifier */
   position_type           posn;        /* position */
   uint32                  unused;      /* should be set to zero */
   char                    cmnt[40];    /* comment */
   uint8                   smbl;        /* symbol id */
   uint8                   dspl;        /* display option */
   float32                 dst;         /* proximity distance (meters) */
   uint8                   color;       /* waypoint color */
   } D107_Wpt_Type;
```

The enumerated values for the “smbl” member of the D107_Wpt_Type are the same as the “smbl” member of the D103_Wpt_Type.

The enumerated values for the “dspl” member of the D107_Wpt_Type are the same as the “dspl” member of the D103_Wpt_Type.

The enumerated values for the “color” member of the D107_Wpt_Type are shown below:

```
enum
  {
  clr_default           = 0,         /* Default waypoint color */
  clr_red               = 1,         /* Red */
  clr_green             = 2,         /* Green */
  clr_blue              = 3          /* Blue */
  };
```

7.4.9 D108_Wpt_Type

```
typedef struct
  {
  uint8                 wpt_class;   /* class (see below) */
  uint8                 color;       /* color (see below) */
  uint8                 dspl;        /* display options (see below) */
  uint8                 attr;        /* attributes (see below) */
  symbol_type           smbl;        /* waypoint symbol */
  uint8                 subclass[18];/* subclass */
  position_type         posn;        /* position */
  float32               alt;         /* altitude in meters */
  float32               dpth;        /* depth in meters */
  float32               dist;        /* proximity distance in meters */
  char                  state[2];    /* state */
  char                  cc[2];       /* country code */
/* char                 ident[];        variable length string */
/* char                 comment[];      waypoint user comment */
/* char                 facility[];     facility name */
/* char                 city[];         city name */
/* char                 addr[];         address number */
/* char                 cross_road[];   intersecting road label */
  } D108_Wpt_Type;
```

The enumerated values for the “wpt_class” member of the D108_Wpt_Type are defined as follows:

```
enum
  {
  user_wpt              = 0x00,      /* user waypoint */
  avtn_apt_wpt          = 0x40,      /* aviation airport waypoint */
  avtn_int_wpt          = 0x41,      /* aviation intersection waypoint */
  avtn_ndb_wpt          = 0x42,      /* aviation NDB waypoint */
  avtn_vor_wpt          = 0x43,      /* aviation VOR waypoint */
  avtn_arwy_wpt         = 0x44,      /* aviation airport runway waypoint */
  avtn_aint_wpt         = 0x45,      /* aviation airport intersection */
  avtn_andb_wpt         = 0x46,      /* aviation airport ndb waypoint */
  map_pnt_wpt           = 0x80,      /* map point waypoint */
  map_area_wpt          = 0x81,      /* map area waypoint */
  map_int_wpt           = 0x82,      /* map intersection waypoint */
  map_adrs_wpt          = 0x83,      /* map address waypoint */
  map_line_wpt          = 0x84,      /* map line waypoint */
  };
```

The “color” member can be one of the following values:

```
enum
  {
  clr_black             = 0,
  clr_dark_red          = 1,
  clr_dark_green        = 2,
  clr_dark_yellow       = 3,
  clr_dark_blue         = 4,
  clr_dark_magenta      = 5,
  clr_dark_cyan         = 6,
  clr_light_gray        = 7,
  clr_dark_gray         = 8,
  clr_red               = 9,
  clr_green             = 10,
  clr_yellow            = 11,
  clr_blue              = 12,
  clr_magenta           = 13,
  clr_cyan              = 14,
  clr_white             = 15,
  clr_default_color     = 255
  };
```

The enumerated values for the "dspl" member of the D108_Wpt_Type are the same as the "dspl" member of the D103_Wpt_Type.

The "attr" member should be set to a value of 0x60.

The "subclass" member of the D108_Wpt_Type is used for map waypoints only, and should be set to 0x0000 0x00000000 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF for other classes of waypoints.

The "alt" and "dpth" members may or may not be supported on a given device. A value of 1.0e25 in either of these fields indicates that this parameter is not supported or is unknown for this waypoint.

The "dist" member is used during the Proximity Waypoint Transfer Protocol only, and should be set to 1.0e25 for other cases.

The "comment" member of the D108_Wpt_Type is used for user waypoints only, and should be an empty string for other waypoint classes.

The "facility" and "city" members are used only for aviation waypoints, and should be empty strings for other waypoint classes.

The "addr" member is only valid for MAP_ADRS_WPT class waypoints and will be an empty string otherwise.

The "cross_road" member is valid only for MAP_INT_WPT class waypoints, and will be an empty string otherwise.

7.4.10 D109_Wpt_Type

```
typedef struct
   {
   uint8               dtyp;           /* data packet type (0x01 for D109) */
   uint8               wpt_class;      /* class */
   uint8               dspl_color;     /* display & color (see below) */
   uint8               attr;           /* attributes (0x70 for D109) */
   symbol_type         smbl;           /* waypoint symbol */
   uint8               subclass[18];/* subclass */
   position_type       posn;           /* position */
   float32             alt;            /* altitude in meters */
   float32             dpth;           /* depth in meters */
   float32             dist;           /* proximity distance in meters */
   char                state[2];       /* state */
   char                cc[2];          /* country code */
   uint32              ete;            /* outbound link ete in seconds */
/* char                ident[];           variable length string */
/* char                comment[];         waypoint user comment */
/* char                facility[];        facility name */
/* char                city[];            city name */
/* char                addr[];            address number */
/* char                cross_road[];      intersecting road label */
   } D109_Wpt_Type;
```

All fields are defined the same as D108_Wpt_Type except as noted below.

dtyp - Data packet type, must be 0x01 for D109_Wpt_Type.

dspl_color - The 'dspl_color' member contains three fields; bits 0-4 specify the color, bits 5-6 specify the waypoint display attribute and bit 7 is unused and must be 0. Color values are as specified for D108_Wpt_Type except that the default value is 0x1f. Display attribute values are as specified for D108_Wpt_Type.

attr - Attribute. Must be 0x70 for D109_Wpt_Type.

ete - Estimated time en route in seconds to next waypoint. Default value is 0xFFFFFFFF.

7.4.11 D110_Wpt_Type

```
typedef struct
  {
  uint8          dtyp;          /* data packet type (0x01 for D110) */
  uint8          wpt_class;     /* class */
  uint8          dspl_color;    /* display & color (see below) */
  uint8          attr;          /* attributes (0x80 for D110) */
  symbol_type    smbl;          /* waypoint symbol */
  uint8          subclass[18];  /* subclass */
  position_type  posn;          /* position */
  float32        alt;           /* altitude in meters */
  float32        dpth;          /* depth in meters */
  float32        dist;          /* proximity distance in meters */
  char           state[2];      /* state */
  char           cc[2];         /* country code */
  uint32         ete;           /* outbound link ete in seconds */
  float32        temp;          /* temperature */
  time_type      time;          /* timestamp */
  uint16         wpt_cat;       /* category membership */
/* char          ident[];       variable length string */
/* char          comment[];     waypoint user comment */
/* char          facility[];    facility name */
/* char          city[];        city name */
/* char          addr[];        address number */
/* char          cross_road[];  intersecting road label */
  } D110_Wpt_Type;
```

All fields are defined the same as D109_Wpt_Type except as noted below.

The valid values for the "wpt_class" member of the D110_Wpt_Type are defined as follows. If an invalid value is received, the value shall be user_wpt.

```
enum
  {
  user_wpt       = 0x00,   /* user waypoint */
  avtn_apt_wpt   = 0x40,   /* aviation airport waypoint */
  avtn_int_wpt   = 0x41,   /* aviation intersection waypoint */
  avtn_ndb_wpt   = 0x42,   /* aviation NDB waypoint */
  avtn_vor_wpt   = 0x43,   /* aviation VOR waypoint */
  avtn_arwy_wpt  = 0x44,   /* aviation airport runway waypoint */
  avtn_aint_wpt  = 0x45,   /* aviation airport intersection */
  avtn_andb_wpt  = 0x46,   /* aviation airport ndb waypoint */
  map_pnt_wpt    = 0x80,   /* map point waypoint */
  map_area_wpt   = 0x81,   /* map area waypoint */
  map_int_wpt    = 0x82,   /* map intersection waypoint */
  map_adrs_wpt   = 0x83,   /* map address waypoint */
  map_line_wpt   = 0x84,   /* map line waypoint */
  };
```

wpt_cat - Waypoint Category. May not be supported by all devices. Default value is 0x0000. This is a bit field that provides category membership information for the waypoint. The waypoint may be a member of up to 16 categories. If a bit is set then the waypoint is a member of the corresponding category. For example, if bits 0 and 4 are set then the waypoint is a member of categories 1 and 5. For more information see section 6.5 on page 12.

temp - Temperature. May not be supported by all devices. A value of 1.0e25 in this field indicates that this parameter is not supported or is unknown for this waypoint.

time - Time. May not be supported by all devices. A value of 0xFFFFFFFF in this field indicates that this parameter is not supported or is unknown for this waypoint.

attr - Attribute. Must be 0x80 for D110_Wpt_Type.

dspl_color - The 'dspl_color' member contains three fields; bits 0-4 specify the color, bits 5-6 specify the waypoint display attribute and bit 7 is unused and must be 0. Valid color values are specified below. If an invalid color value is received, the value shall be Black. Valid display attribute values are as shown below. If an invalid display attribute value is received, the value shall be Name.

```
enum
   {
   clr_Black               = 0,
   clr_Dark_Red            = 1,
   clr_Dark_Green          = 2,
   clr_Dark_Yellow         = 3,
   clr_Dark_Blue           = 4,
   clr_Dark_Magenta        = 5,
   clr_Dark_Cyan           = 6,
   clr_Light_Gray          = 7,
   clr_Dark_Gray           = 8,
   clr_Red                 = 9,
   clr_Green               = 10,
   clr_Yellow              = 11,
   clr_Blue                = 12,
   clr_Magenta             = 13,
   clr_Cyan                = 14,
   clr_White               = 15,
   clr_Transparent         = 16
   };
```

```
enum
   {
   dspl_Smbl_Name          = 0,          /* Display symbol with waypoint name */
   dspl_Smbl_Only          = 1,          /* Display symbol by itself */
   dspl_Smbl_Comment       = 2           /* Display symbol with comment */
   };
```

posn - Position. If a D110 waypoint is received that contains a value in the lat field of the posn field that is greater than 2^30 or less than -2^30, then that waypoint shall be rejected.

7.4.12 D120_Wpt_Cat_Type

```
typedef struct
   {
   char                    name[17];     /* category name */
   } D120_Wpt_Cat_Type;
```

The name field contains a null-terminated string with a maximum length of 16 consecutive non-null characters. If a D120 waypoint category is received that contains a string with more than 16 consecutive non-null characters then that name should be truncated to the first 16 characters and then null terminated. If a D120 waypoint category is received with a null in the first character of the name field then that packet should not be processed.

7.4.13 D150_Wpt_Type

```
typedef struct
   {
   char                    ident[6];     /* identifier */
   char                    cc[2];        /* country code */
   uint8                   wpt_class;    /* class */
   position_type           posn;         /* position */
   sint16                  alt;          /* altitude (meters) */
   char                    city[24];     /* city */
   char                    state[2];     /* state */
   char                    name[30];     /* facility name */
   char                    cmnt[40];     /* comment */
   } D150_Wpt_Type;
```

The enumerated values for the “wpt_class” member of the D150_Wpt_Type are shown below:

```
enum
  {
  apt_wpt_class          = 0,        /* airport waypoint class */
  int_wpt_class          = 1,        /* intersection waypoint class */
  ndb_wpt_class          = 2,        /* NDB waypoint class */
  vor_wpt_class          = 3,        /* VOR waypoint class */
  usr_wpt_class          = 4,        /* user defined waypoint class */
  rwy_wpt_class          = 5,        /* airport runway threshold waypoint class */
  aint_wpt_class         = 6,        /* airport intersection waypoint class */
  locked_wpt_class       = 7         /* locked waypoint class */
  };
```

The “locked_wpt_class” code indicates that a route within a device contains an aviation database waypoint that the device could not find in its aviation database (presumably because the aviation database was updated to a newer version). The host should never send the “locked_wpt_class” code to the device.

The “city,” “state,” “name,” and “cc” members are invalid when the “wpt_class” member is equal to usr_wpt_class. The “alt” member is valid only when the “wpt_class” member is equal to apt_wpt_class.

7.4.14 D151_Wpt_Type

```
typedef struct
  {
  char                   ident[6];   /* identifier */
  position_type          posn;       /* position */
  uint32                 unused;     /* should be set to zero */
  char                   cmnt[40];   /* comment */
  float32                dst;        /* proximity distance (meters) */
  char                   name[30];   /* facility name */
  char                   city[24];   /* city */
  char                   state[2];   /* state */
  sint16                 alt;        /* altitude (meters) */
  char                   cc[2];      /* country code */
  char                   unused2;    /* should be set to zero */
  uint8                  wpt_class;  /* class */
  } D151_Wpt_Type;
```

The enumerated values for the “wpt_class” member of the D151_Wpt_Type are shown below:

```
enum
  {
  apt_wpt_class          = 0,        /* airport waypoint class */
  vor_wpt_class          = 1,        /* VOR waypoint class */
  usr_wpt_class          = 2,        /* user defined waypoint class */
  locked_wpt_class       = 3         /* locked waypoint class */
  };
```

The “locked_wpt_class” code indicates that a route within a device contains an aviation database waypoint that the device could not find in its aviation database (presumably because the aviation database was updated to a newer version). The host should never send the “locked_wpt_class” code to the device.

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

The “city,” “state,” “name,” and “cc” members are invalid when the “wpt_class” member is equal to usr_wpt_class. The “alt” member is valid only when the “wpt_class” member is equal to apt_wpt_class.

7.4.15 D152_Wpt_Type

```
typedef struct
  {
  char                    ident[6];     /* identifier */
  position_type           posn;         /* position */
  uint32                  unused;       /* should be set to zero */
  char                    cmnt[40];     /* comment */
  float32                 dst;          /* proximity distance (meters) */
  char                    name[30];     /* facility name */
  char                    city[24];     /* city */
  char                    state[2];     /* state */
  sint16                  alt;          /* altitude (meters) */
  char                    cc[2];        /* country code */
  uint8                   unused2;      /* should be set to zero */
  uint8                   wpt_class;    /* class */
  } D152_Wpt_Type;
```

The enumerated values for the "wpt_class" member of the D152_Wpt_Type are shown below:

```
enum
  {
  apt_wpt_class           = 0,          /* airport waypoint class */
  int_wpt_class           = 1,          /* intersection waypoint class */
  ndb_wpt_class           = 2,          /* NDB waypoint class */
  vor_wpt_class           = 3,          /* VOR waypoint class */
  usr_wpt_class           = 4,          /* user defined waypoint class */
  locked_wpt_class        = 5           /* locked waypoint class */
  };
```

The "locked_wpt_class" code indicates that a route within a device contains an aviation database waypoint that the device could not find in its aviation database (presumably because the aviation database was updated to a newer version). The host should never send the "locked_wpt_class" code to the device.

The "dst" member is valid only during the Proximity Waypoint Transfer Protocol.

The "city," "state," "name," and "cc" members are invalid when the "wpt_class" member is equal to usr_wpt_class. The "alt" member is valid only when the "wpt_class" member is equal to apt_wpt_class.

7.4.16 D154_Wpt_Type

```
typedef struct
  {
  char                    ident[6];     /* identifier */
  position_type           posn;         /* position */
  uint32                  unused;       /* should be set to zero */
  char                    cmnt[40];     /* comment */
  float32                 dst;          /* proximity distance (meters) */
  char                    name[30];     /* facility name */
  char                    city[24];     /* city */
  char                    state[2];     /* state */
  sint16                  alt;          /* altitude (meters) */
  char                    cc[2];        /* country code */
  uint8                   unused2;      /* should be set to zero */
  uint8                   wpt_class;    /* class */
  symbol_type             smbl;         /* symbol id */
  } D154_Wpt_Type;
```

The enumerated values for the "wpt_class" member of the D154_Wpt_Type are shown below:

```
enum
  {
  apt_wpt_class         = 0,        /* airport waypoint class */
  int_wpt_class         = 1,        /* intersection waypoint class */
  ndb_wpt_class         = 2,        /* NDB waypoint class */
  vor_wpt_class         = 3,        /* VOR waypoint class */
  usr_wpt_class         = 4,        /* user defined waypoint class */
  rwy_wpt_class         = 5,        /* airport runway threshold waypoint class */
  aint_wpt_class        = 6,        /* airport intersection waypoint class */
  andb_wpt_class        = 7,        /* airport NDB waypoint class */
  sym_wpt_class         = 8,        /* user defined symbol-only waypoint class */
  locked_wpt_class      = 9         /* locked waypoint class */
  };
```

The “locked_wpt_class” code indicates that a route within a device contains an aviation database waypoint that the device could not find in its aviation database (presumably because the aviation database was updated to a newer version). The host should never send the “locked_wpt_class” code to the device.

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

The “city,” “state,” “name,” and “cc” members are invalid when the “wpt_class” member is equal to usr_wpt_class or sym_wpt_class. The “alt” member is valid only when the “wpt_class” member is equal to apt_wpt_class.

7.4.17 D155_Wpt_Type

```
typedef struct
  {
  char                  ident[6];   /* identifier */
  position_type         posn;       /* position */
  uint32                unused;     /* should be set to zero */
  char                  cmnt[40];   /* comment */
  float32               dst;        /* proximity distance (meters) */
  char                  name[30];   /* facility name */
  char                  city[24];   /* city */
  char                  state[2];   /* state */
  sint16                alt;        /* altitude (meters) */
  char                  cc[2];      /* country code */
  uint8                 unused2;    /* should be set to zero */
  uint8                 wpt_class;  /* class */
  symbol_type           smbl;       /* symbol id */
  uint8                 dspl;       /* display option */
  } D155_Wpt_Type;
```

The enumerated values for the “dspl” member of the D155_Wpt_Type are shown below:

```
enum
  {
  dspl_smbl_only        = 1,        /* Display symbol by itself */
  dspl_smbl_name        = 3,        /* Display symbol with waypoint name */
  dspl_smbl_cmnt        = 5,        /* Display symbol with comment */
  };
```

The enumerated values for the “wpt_class” member of the D155_Wpt_Type are shown below:

```
enum
  {
  apt_wpt_class         = 0,        /* airport waypoint class */
  int_wpt_class         = 1,        /* intersection waypoint class */
  ndb_wpt_class         = 2,        /* NDB waypoint class */
  vor_wpt_class         = 3,        /* VOR waypoint class */
  usr_wpt_class         = 4,        /* user defined waypoint class */
  locked_wpt_class      = 5         /* locked waypoint class */
  };
```

The “locked_wpt_class” code indicates that a route within a device contains an aviation database waypoint that the device could not find in its aviation database (presumably because the aviation database was updated to a newer version). The host should never send the “locked_wpt_class” code to the device.

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

The “city,” “state,” “name,” and “cc” members are invalid when the “wpt_class” member is equal to usr_wpt_class. The “alt” member is valid only when the “wpt_class” member is equal to apt_wpt_class.

7.4.18 D200_Rte_Hdr_Type

```
typedef uint8 D200_Rte_Hdr_Type;        /* route number */
```

The route number contained in the D200_Rte_Hdr_Type must be unique for each route.

7.4.19 D201_Rte_Hdr_Type

```
typedef struct
   {
   uint8                  nmbr;         /* route number */
   char                   cmnt[20];     /* comment */
   } D201_Rte_Hdr_Type;
```

The “nmbr” member must be unique for each route. Some devices require a unique “cmnt” for each route, and other devices do not. There is no mechanism available for the host to determine whether a device requires a unique “cmnt”, and the host must be prepared to receive unique or non-unique “cmnt” from the device.

7.4.20 D202_Rte_Hdr_Type

```
typedef struct
   {
/* char                   rte_ident[];     variable length string */
   } D202_Rte_Hdr_Type;
```

7.4.21 D210_Rte_Link_Type

```
typedef struct
   {
   uint16                 class;        /* link class; see below */
   uint8                  subclass[18]; /* subclass */
/* char                   ident[];        variable length string */
   };
```

The “class” member can be one of the following values:

```
enum
   {
   line                   = 0,
   link                   = 1,
   net                    = 2,
   direct                 = 3,
   snap                   = 0xFF
   };
```

The “ident” member has a maximum length of 51 characters, including the terminating NULL.

If “class” is set to “direct” or “snap”, subclass should be set to its default value of 0x0000 0x00000000 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF.

7.4.22 D300_Trk_Point_Type

```
typedef struct
   {
   position_type            posn;         /* position */
   time_type                time;         /* time */
   bool                     new_trk;      /* new track segment? */
   } D300_Trk_Point_Type;
```

The "time" member indicates the time at which the track log point was recorded.

When true, the "new_trk" member indicates that the track log point marks the beginning of a new track log segment.

7.4.23 D301_Trk_Point_Type

```
typedef struct
   {
   position_type            posn;         /* position */
   time_type                time;         /* time */
   float32                  alt;          /* altitude in meters */
   float32                  dpth;         /* depth in meters */
   bool                     new_trk;      /* new track segment? */
   } D301_Trk_Point_Type;
```

The "time" member indicates the time at which the track log point was recorded.

The 'alt' and 'dpth' members may or may not be supported on a given device. A value of 1.0e25 in either of these fields indicates that this parameter is not supported or is unknown for this track point.

When true, the "new_trk" member indicates that the track log point marks the beginning of a new track log segment.

7.4.24 D302_Trk_Point_Type

```
typedef struct
   {
   position_type            posn;         /* position */
   time_type                time;         /* time */
   float32                  alt;          /* altitude in meters */
   float32                  dpth;         /* depth in meters */
   float32                  temp;         /* temp in degrees C */
   bool                     new_trk;      /* new track segment? */
   } D302_Trk_Point_Type;
```

All fields are defined the same as D301_Trk_Point_Type except as noted below.

temp - Temperature. May not be supported by all devices. A value of 1.0e25 in this field indicates that this parameter is not supported or is unknown for this track point.

7.4.25 D310_Trk_Hdr_Type

```
typedef struct
   {
   bool                     dspl;         /* display on the map? */
   uint8                    color;        /* color (same as D108) */
/* char                     trk_ident[];     null-terminated string */
   } D310_Trk_Hdr_Type;
```

The 'trk_ident' member has a maximum length of 51 characters including the terminating NULL.

7.4.26 D311_Trk_Hdr_Type

```
typedef struct
   {
   uint16                  index;        /* unique among all tracks received from device
*/
   } D311_Trk_Hdr_Type;
```

7.4.27 D312_Trk_Hdr_Type

```
typedef struct
   {
   bool                    dspl;         /* display on the map? */
   uint8                   color;        /* color (same as D110) */
/* char                    trk_ident[];     null-terminated string */
   } D312_Trk_Hdr_Type;
```

The 'trk_ident' member has a maximum length of 51 characters including the terminating NULL.

7.4.28 D400_Prx_Wpt_Type

```
typedef struct
   {
   D100_Wpt_Type           wpt;          /* waypoint */
   float32                 dst;          /* proximity distance (meters) */
   } D400_Prx_Wpt_Type;
```

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.29 D403_Prx_Wpt_Type

```
typedef struct
   {
   D103_Wpt_Type           wpt;          /* waypoint */
   float32                 dst;          /* proximity distance (meters) */
   } D403_Prx_Wpt_Type;
```

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.30 D450_Prx_Wpt_Type

```
typedef struct
   {
   int                     idx;          /* proximity index */
   D150_Wpt_Type           wpt;          /* waypoint */
   float32                 dst;          /* proximity distance (meters) */
   } D450_Prx_Wpt_Type;
```

The “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

7.4.31 D500_Almanac_Type

```
typedef struct
  {
  uint16          wn;        /* week number (weeks) */
  float32         toa;       /* almanac data reference time (s) */
  float32         af0;       /* clock correction coefficient (s) */
  float32         af1;       /* clock correction coefficient (s/s) */
  float32         e;         /* eccentricity (-) */
  float32         sqrta;     /* square root of semi-major axis (a)(m**1/2) */
  float32         m0;        /* mean anomaly at reference time (r) */
  float32         w;         /* argument of perigee (r) */
  float32         omg0;      /* right ascension (r) */
  float32         odot;      /* rate of right ascension (r/s) */
  float32         i;         /* inclination angle (r) */
  } D500_Almanac_Type;
```

7.4.32 D501_Almanac_Type

```
typedef struct
  {
  uint16          wn;        /* week number (weeks) */
  float32         toa;       /* almanac data reference time (s) */
  float32         af0;       /* clock correction coefficient (s) */
  float32         af1;       /* clock correction coefficient (s/s) */
  float32         e;         /* eccentricity (-) */
  float32         sqrta;     /* square root of semi-major axis (a)(m**1/2) */
  float32         m0;        /* mean anomaly at reference time (r) */
  float32         w;         /* argument of perigee (r) */
  float32         omg0;      /* right ascension (r) */
  float32         odot;      /* rate of right ascension (r/s) */
  float32         i;         /* inclination angle (r) */
  uint8           hlth;      /* almanac health */
  } D501_Almanac_Type;
```

7.4.33 D550_Almanac_Type

```
typedef struct
  {
  uint8           svid;      /* satellite id */
  uint16          wn;        /* week number (weeks) */
  float32         toa;       /* almanac data reference time (s) */
  float32         af0;       /* clock correction coefficient (s) */
  float32         af1;       /* clock correction coefficient (s/s) */
  float32         e;         /* eccentricity (-) */
  float32         sqrta;     /* square root of semi-major axis (a)(m**1/2) */
  float32         m0;        /* mean anomaly at reference time (r) */
  float32         w;         /* argument of perigee (r) */
  float32         omg0;      /* right ascension (r) */
  float32         odot;      /* rate of right ascension (r/s) */
  float32         i;         /* inclination angle (r) */
  } D550_Almanac_Type;
```

The “svid” member identifies a satellite in the GPS constellation as follows: PRN-01 through PRN-32 are indicated by “svid” equal to 0 through 31, respectively.

7.4.34 D551_Almanac_Type

```
typedef struct
  {
  uint8              svid;       /* satellite id */
  uint16             wn;         /* week number (weeks) */
  float32            toa;        /* almanac data reference time (s) */
  float32            af0;        /* clock correction coefficient (s) */
  float32            af1;        /* clock correction coefficient (s/s) */
  float32            e;          /* eccentricity (-) */
  float32            sqrta;      /* square root of semi-major axis (a)(m**1/2) */
  float32            m0;         /* mean anomaly at reference time (r) */
  float32            w;          /* argument of perigee (r) */
  float32            omg0;       /* right ascension (r) */
  float32            odot;       /* rate of right ascension (r/s) */
  float32            i;          /* inclination angle (r) */
  uint8              hlth;       /* almanac health bits 17:24 (coded) */
  } D551_Almanac_Type;
```

The “svid” member identifies a satellite in the GPS constellation as follows: PRN-01 through PRN-32 are indicated by “svid” equal to 0 through 31, respectively.

7.4.35 D600_Date_Time_Type

```
typedef struct
  {
  uint8              month;      /* month  (1-12) */
  uint8              day;        /* day    (1-31) */
  uint16             year;       /* year   (1990 means 1990) */
  uint16             hour;       /* hour   (0-23) */
  uint8              minute;     /* minute (0-59) */
  uint8              second;     /* second (0-59) */
  } D600_Date_Time_Type;
```

The D600_Date_Time_Type contains the UTC date and UTC time.

7.4.36 D650_FlightBook_Record_Type

```
typedef struct
  {
  time_type          takeoff_time;/* Time flight started */
  time_type          landing_time;/* Time flight ended */
  position_type      takeoff_posn;/* Takeoff lat/lon */
  position_type      landing_posn;/* Takeoff lat/lon */
  uint32             night_time;  /* Seconds flown in night time conditions */
  uint32             num_landings;/* Number of landings during the flight */
  float32            max_speed;   /* Max velocity during flight (meters/sec) */
  float32            max_alt;     /* Max altitude above WGS84 ellipsoid (meters)*/
  float32            distance;    /* Distance of flight (meters) */
  bool               cross_country_flag; /* Flight met cross country criteria */
/* char              departure_name[]; Name of airport        <= 31 bytes */
/* char              departure_ident[]; ID of airport         <= 11 bytes */
/* char              arrival_name[]; Name of airport          <= 31 bytes */
/* char              arrival_ident[];ID of airport           <= 11 bytes */
/* char              ac_id[];        N Number of airplane        <= 11 bytes */
  } D650_Flight_Book_Record_Type;
```

7.4.37 D700_Position_Type

```
typedef radian_position_type D700_Position_Type;
```

7.4.38 D800_Pvt_Data_Type

```
typedef struct
  {
  float32              alt;        /* altitude above WGS 84 ellipsoid (meters) */
  float32              epe;        /* estimated position error, 2 sigma (meters) */
  float32              eph;        /* epe, but horizontal only (meters) */
  float32              epv;        /* epe, but vertical only (meters) */
  uint16               fix;        /* type of position fix */
  float64              tow;        /* time of week (seconds) */
  radian_position_type posn;       /* latitude and longitude (radians) */
  float32              east;       /* velocity east  (meters/second) */
  float32              north;      /* velocity north (meters/second) */
  float32              up;         /* velocity up    (meters/second) */
  float32              msl_hght;   /* height of WGS84 ellipsoid above MSL(meters)*/
  sint16               leap_scnds; /* difference between GPS and UTC (seconds) */
  uint32               wn_days;    /* week number days */
  } D800_Pvt_Data_Type;
```

The “alt” parameter provides the altitude above the WGS 84 ellipsoid. To find the altitude above mean sea level, add “msl_hght” to “alt” (“msl_hght” gives the height of the WGS 84 ellipsoid above mean sea level at the current position).

The “tow” parameter provides the number of seconds (excluding leap seconds) since the beginning of the current week, which begins on Sunday at 12:00 AM (i.e., midnight Saturday night-Sunday morning). The “tow” parameter is based on Universal Coordinated Time (UTC), except UTC is periodically corrected for leap seconds while “tow” is not corrected for leap seconds. To find UTC, subtract “leap_scnds” from “tow.” Since this may cause a negative result for the first few seconds of the week (i.e., when “tow” is less than “leap_scnds”), care must be taken to properly translate this negative result to a positive time value in the previous day. Also, since “tow” is a floating point number and may contain fractional seconds, care must be taken to properly round off when using “tow” in integer conversions and calculations.

The “wn_days” parameter provides the number of days that have occurred from UTC December 31st, 1989 to the beginning of the current week (thus, “wn_days” always represents a Sunday). To find the total number of days that have occurred from UTC December 31st, 1989 to the current day, add “wn_days” to the number of days that have occurred in the current week (as calculated from the “tow” parameter).

The default enumerated values for the “fix” member of the D800_Pvt_Data_Type are shown below. It is important for the host to inspect this value to ensure that other data members in the D800_Pvt_Data_Type are valid. No indication is given as to whether the device is in simulator mode versus having an actual position fix.

```
enum
  {
  unusable             = 0,        /* failed integrity check */
  invalid              = 1,        /* invalid or unavailable */
  2D                   = 2,        /* two dimensional */
  3D                   = 3,        /* three dimensional */
  2D_diff              = 4,        /* two dimensional differential */
  3D_diff              = 5         /* three dimensional differential */
  };
```

Older software versions in certain devices use slightly different enumerated values for fix. The list of devices and the last version of software in which these different values are used is:

Device	Last SW Version
eMap	2.64
GPSMAP 162	2.62
GPSMAP 295	2.19
eTrex	2.10
eTrex Summit	2.07
StreetPilot III	2.10
eTrex Japanese	2.10
eTrex Venture/Mariner	2.20
eTrex Europe	2.03
GPS 152	2.01
eTrex Chinese	2.01
eTrex Vista	2.12
eTrex Summit Japanese	2.01
eTrex Summit	2.24
eTrex GolfLogix	2.49

The enumerated values for these device software versions is one more than the default:

```
enum
   {
   unusable                 = 1,         /* failed integrity check */
   invalid                  = 2,         /* invalid or unavailable */
   2D                       = 3,         /* two dimensional */
   3D                       = 4,         /* three dimensional */
   2D_diff                  = 5,         /* two dimensional differential */
   3D_diff                  = 6          /* three dimensional differential */
   };
```

7.4.39 D906_Lap_Type

```
typedef struct
   {
   time_type                start_time;
   uint32                   total_time;  /* In hundredths of a second */
   float32                  total_distance; /* In meters */
   position_type            begin;       /* Invalid if both lat and lon are 0x7FFFFFFF */
   position_type            end;         /* Invalid if both lat and lon are 0x7FFFFFFF */
   uint16                   calories;
   uint8                    track index; /* See below */
   uint8                    unused;      /* Unused. Set to 0. */
   } D906_Lap Type;
```

Possible values for the track_index member of the D906_Lap_Type are as follows:

Value	**Meaning**
0 - 252	The lap is the last in its run. The track index is valid and can be used to lookup the track and associate it with the run.
253 - 254	The lap is the last in its run; however, the run has no associated track.
255	The lap is not the last in its run. Or, the lap is the last lap received so it is the last lap in its run. The track for the run is any track not already associated with a run.

8 Appendixes

8.1 Device Product IDs

The table below provides the Product ID numbers for many Garmin devices.

Table 27 – Product IDs

Product Name	ID
GNC 250	52
GNC 250 XL	64
GNC 300	33
GNC 300 XL	98
GPS 12	77
GPS 12	87
GPS 12	96
GPS 12 XL	77
GPS 12 XL	96
GPS 12 XL Chinese	106
GPS 12 XL Japanese	105
GPS 120	47
GPS 120 Chinese	55
GPS 120 XL	74
GPS 125 Sounder	61
GPS 126	95
GPS 126 Chinese	100
GPS 128	95
GPS 128 Chinese	100
GPS 150	20
GPS 150 XL	64
GPS 155	34
GPS 155 XL	98
GPS 165	34
GPS 38	41
GPS 38 Chinese	56
GPS 38 Japanese	62
GPS 40	31
GPS 40	41
GPS 40 Chinese	56
GPS 40 Japanese	62
GPS 45	31
GPS 45	41
GPS 45 Chinese	56
GPS 45 XL	41
GPS 48	96
GPS 50	7

Product Name	ID
GPS 55	14
GPS 55 AVD	15
GPS 65	18
GPS 75	13
GPS 75	23
GPS 75	42
GPS 85	25
GPS 89	39
GPS 90	45
GPS 92	112
GPS 95	24
GPS 95	35
GPS 95 AVD	22
GPS 95 AVD	36
GPS 95 XL	36
GPS II	59
GPS II Plus	73
GPS II Plus	97
GPS III	72
GPS III Pilot	71
GPSCOM 170	50
GPSCOM 190	53
GPSMAP 130	49
GPSMAP 130 Chinese	76
GPSMAP 135 Sounder	49
GPSMAP 175	49
GPSMAP 195	48
GPSMAP 205	29
GPSMAP 205	44
GPSMAP 210	29
GPSMAP 215	88
GPSMAP 220	29
GPSMAP 225	88
GPSMAP 230	49
GPSMAP 230 Chinese	76
GPSMAP 235 Sounder	49

8.2 Device Protocol Capabilities

Table 28 below provides the protocol capabilities of many devices that do not implement the Protocol Capability Protocol (see section 6.2 on page 9). Column 1 contains the applicable Product ID number, and Column 2 contains the applicable software version number. The remaining columns show the device-specific protocol IDs and data type IDs for the types of protocols indicated. Within these remaining columns, protocol IDs are prefixed with P, L, or A (Physical, Link, or Application) and data type IDs are prefixed with D.

The presence of a device in the table indicates that the device did not originally implement the Protocol Capabilities Protocol (A001). However, if the host detects that one of these devices now provides Protocol Capabilities Protocol data (due to a new version of software loaded in the device), then Protocol Capabilities Protocol data shall take precedence over the data provided in the table below.

The following protocols are omitted from the table because all devices in the table implement them:

A000	Product Data Protocol
A600	Date and Time Initialization Protocol
A700	Position Initialization Protocol

All devices in the table use the D600 data type in conjunction with the A600 protocol; similarly, all devices in the table use the D700 data type in conjunction with the A700 protocol. The A800/D800 protocol and data type are omitted from the table because none of the devices in the table implements PVT Data transfer.

Note: all numbers are in decimal format.

Table 28 – Device Protocol Capabilities

ID	Version	Link	Command	Waypoint	Route	Track	Proximity	Almanac
7	All	L001	A010	A100 D100	A200 D200 D100			A500 D500
25	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
13	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
14	All	L001	A010	A100 D100	A200 D200 D100		A400 D400	A500 D500
15	All	L001	A010	A100 D151	A200 D200 D151		A400 D151	A500 D500
18	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
20	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D550
22	All	L001	A010	A100 D152	A200 D200 D152	A300 D300	A400 D152	A500 D500
23	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
24	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
29	< 4.00	L001	A010	A100 D101	A200 D201 D101	A300 D300	A400 D101	A500 D500
29	>= 4.00	L001	A010	A100 D102	A200 D201 D102	A300 D300	A400 D102	A500 D500
31	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
33	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D550
34	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D550
35	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
36	< 3.00	L001	A010	A100 D152	A200 D200 D152	A300 D300	A400 D152	A500 D500

ID	Version	Link	Command	Waypoint	Route	Track	Proximity	Almanac
36	>= 3.00	L001	A010	A100 D152	A200 D200 D152	A300 D300		A500 D500
39	All	L001	A010	A100 D151	A200 D201 D151	A300 D300		A500 D500
41	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
42	All	L001	A010	A100 D100	A200 D200 D100	A300 D300	A400 D400	A500 D500
44	All	L001	A010	A100 D101	A200 D201 D101	A300 D300	A400 D101	A500 D500
45	All	L001	A010	A100 D152	A200 D201 D152	A300 D300		A500 D500
47	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
48	All	L001	A010	A100 D154	A200 D201 D154	A300 D300		A500 D501
49	All	L001	A010	A100 D102	A200 D201 D102	A300 D300	A400 D102	A500 D501
50	All	L001	A010	A100 D152	A200 D201 D152	A300 D300		A500 D501
52	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D550
53	All	L001	A010	A100 D152	A200 D201 D152	A300 D300		A500 D501
55	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
56	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
59	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
61	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
62	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
64	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D551

ID	Version	Link	Command	Waypoint	Route	Track	Proximity	Almanac
71	All	L001	A010	A100 D155	A200 D201 D155	A300 D300		A500 D501
72	All	L001	A010	A100 D104	A200 D201 D104	A300 D300		A500 D501
73	All	L001	A010	A100 D103	A200 D201 D103	A300 D300		A500 D501
74	All	L001	A010	A100 D100	A200 D201 D100	A300 D300		A500 D500
76	All	L001	A010	A100 D102	A200 D201 D102	A300 D300	A400 D102	A500 D501
77	< 3.01	L001	A010	A100 D100	A200 D201 D100	A300 D300	A400 D400	A500 D501
77	>= 3.01 < 3.50	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
77	>= 3.50 < 3.61	L001	A010	A100 D103	A200 D201 D103	A300 D300		A500 D501
77	>= 3.61	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
87	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
88	All	L001	A010	A100 D102	A200 D201 D102	A300 D300	A400 D102	A500 D501
95	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
96	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
97	All	L001	A010	A100 D103	A200 D201 D103	A300 D300		A500 D501
98	All	L002	A011	A100 D150	A200 D201 D150		A400 D450	A500 D551
100	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
105	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501
106	All	L001	A010	A100 D103	A200 D201 D103	A300 D300	A400 D403	A500 D501

ID	Version	Link	Command	Waypoint	Route	Track	Proximity	Almanac
112	All	L001	A010	A100 D152	A200 D201 D152	A300 D300		A500 D501

8.3 Frequently Asked Questions

8.3.1 Hexadecimal vs. Decimal Numbers

Q: Why doesn't the document contain hexadecimal numbers?

A: Having both decimal and hexadecimal numbers introduces dual-maintenance, which is twice the work and prone to errors. Therefore, we chose to use a single numbering system. We chose decimal because it made the overall document easier to understand.

8.3.2 Length of Received Data Packet

Q: Should my program look at the length of an incoming packet to detect which waypoint format is being sent from the device?

A: Prior to having a definitive interface specification, this was probably the best approach. However, now you should follow the recommendations of the specification and use the Protocol Capabilities Protocol (see section 6.2 on page 9) or Table 28 on page 47 to explicitly determine the waypoint format. Validating data based on length is undesirable because: 1) it doesn't validate the integrity of the data (this is done at the link layer using a checksum); and 2) there is some possibility that the device will transmit a few extra bytes at the end of the data, which would invalidate an otherwise valid packet (you can safely ignore the extra bytes).

8.3.3 Waypoint Creation Date

Q: Isn't the "unused" uint32 in waypoint formats really the date of waypoint creation?

A: Only a few of our very early devices used this field for creation date. All other devices treat it as "unused." Your program should ignore this field when receiving and set it to zero when transmitting.

8.3.4 Almanac Data Parameters

Q: What is meaning of the almanac data parameters such as wn, toa, af0, etc.?

A: No definitions for these parameters are given other than what is provided in the comments. In most cases, a program should simply upload and download this data. Otherwise, the comments should suffice for most applications.

8.3.5 Example Code

Q: Where can I find example code (e.g., for converting time and position formats)?

A: We currently don't have the resources to provide this information.

8.3.6 Sample Data Transfer Dumps

Q: Where can I find some sample data transfer dumps?

A: We currently don't have the resources to provide this information.

8.3.7 Additional Tables

Q: Why doesn't the document contain additional tables (e.g., an additional table in Section 8.1 sorted by Product ID)?

A: We believe the document contains all the necessary information with minimal duplication. Additional sorting may be performed by the copy/pasting the data into your favorite spreadsheet.

8.3.8 Software Versions

Q: Why doesn't Table 27 include an indication of software version?

A: We currently don't have the resources to provide this information. The purpose of the table is to allow you to determine the Product IDs for the devices you wish to support. For example, to support a GPS 12 you must support Product IDs 77, 87, and 96 and their associated protocols from Table 28.