# GARMIN GPS Interface Specification

May 6, 1998

Drawing Number: 001-00063-00 Rev. 1
File Type: Microsoft Word 97
Archive File: 00100063.001

### Notice:



### WARNING:

All companies and individuals accessing the GPS Interface are advised to ensure the correctness of their GPS Interface software and to avoid the use of undocumented GPS 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 GPS.

### Technical Support is not Provided:

GARMIN Corporation cannot provide technical support for questions relating to the GPS 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

# 1. Introduction

## 1.1. Overview

This document describes the GARMIN GPS Interface, which is used to communicate with a GARMIN GPS product. The GPS 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 GPS products are identified.

## 1.2. Definition of Terms

In this document, "GPS" means the GPS device, and "Host" means the device communicating with the GPS (usually a Personal Computer). The term "device" means either the GPS or the Host.

## 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 product-specific data types are found in Section 7, Data Types, on page 26. Data types having limited scope are specified in earlier sections throughout this document (usually in the same section in which they are introduced).

# 2. Protocol Layers

The protocols used in the GARMIN GPS Interface are arranged in the following three 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 GPS. These protocols are described in more detail later in this document.

# 3. Physical Protocols

## 3.1. P000 – Default Physical Protocol

The default Physical protocol is based on RS-232. The voltage characteristics are compatible with most Host devices; however, the GPS 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 GPS voltage characteristics are compatible with most Host devices 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. Provisions are made to support other Physical protocols (primarily higher baud rates), but each GPS product will always operate with the default Physical protocol after power up.

The mechanical characteristics vary among GARMIN products; most products 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.

# 4. Link Protocols

## 4.1. L000 – Basic Link Protocol

All GPS products implement the Basic Link Protocol. Its primary purpose is to facilitate initial communication between Host and GPS using the Product Data Protocol (see Section 6.1 on page 13), which allows the Host to determine which type of GPS is connected. Using this knowledge, the Host can then determine which product-specific Link protocol to use for all other communication with the GPS.

### 4.1.1. 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 diagram shows the format of a packet:

| 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) |

### 4.1.2. 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.

### 4.1.3. 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. 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 a 16-bit integer in their packet data to indicate the Packet ID of the acknowledged packet.

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.

Some GPS products may send NAK packets during communication timeout conditions. For example, when the GPS 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 GPS (perhaps due to cable disconnection during the middle of a protocol sequence). Not all GPS products 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.

### 4.1.4. Basic Packet IDs

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

```
enum
    {
    Pid_Ack_Byte          =   6,
    Pid_Nak_Byte          =  21,
    Pid_Protocol_Array    = 253,        /* may not be implemented in all products */
    Pid_Product_Rqst      = 254,
    Pid_Product_Data      = 255
    };
```

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.

## 4.2. L001 – Link Protocol 1

This Link protocol is used for the majority of GPS products (see Section 8.2, GPS Product Protocol Capabilities, on page 43). 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
    };
```

## 4.3. L002 – Link Protocol 2

This Link protocol is used mainly for panel-mounted aviation GPS products (see Section 8.2, GPS Product Protocol Capabilities, on page 43). 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_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 products 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 products 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 products are able to report which protocols they support (see Section 6.2, A001 – Protocol Capability Protocol, on page 14). In all other cases, the Host must contain a lookup table to determine which protocols to use with which product types (see Section 8.2, GPS Product Protocol Capabilities, on page 43).

## 5.1. 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:

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_First | First_Data_Type |
| 1 | Device1 → Device2 | Pid_Second | ignored |
| 2 | Device1 → Device2 | Pid_Third | <D0> |
| 3 | Device1 ← Device2 | Pid_Fourth | <D1> |
| 4 | Device1 ← Device2 | 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., GPS to Host) is the same as the protocol for transfers in the other direction (e.g., Host to GPS). For symmetric protocols, either the GPS 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 GPS 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 4, Link Protocols, on page 8). The last column shows the Packet Data Type.

## 5.2. 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 product-specific. In the example above, there are three product-specific data types (<D0>, <D1>, and <D2>).

These product-specific data types must be determined dynamically by the Host depending on which type of GPS is currently connected. For older products, this determination is made through the use of a lookup table within the Host (see Section 8.2, GPS Product Protocol Capabilities, on page 43), however, newer GPS products are able to dynamically report their protocols and data types (see Section 6.2, A001 – Protocol Capability Protocol, on page 14).

## 5.3. 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:

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| … | … | … | … |
| n-1 | Device1 → 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, Device Command Protocols, on page 16).

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, Device Command Protocols, on page 16). As a result, the actual Command_Id_Type value depends on which Device Command protocol is implemented by the GPS. 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.

### 5.3.1. 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 int Records_Type;
```

## 5.4. GPS Overwriting of Identically-Named Data

When receiving data from the Host, the GPS 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, it will overwrite the waypoint named XYZ that was previously stored in GPS memory. No warning is sent from the GPS prior to overwriting identically-named data.

# 6. Application Protocols

## 6.1. A000 – Product Data Protocol

All GPS products are required to implement the Product Data Protocol using the default physical and basic link protocols described earlier in this document (i.e., the default RS-232 settings and the default packet format). The Product Data Protocol is used to query the GPS to find out its Product ID, which is then used by the Host to determine which data transfer protocols are supported by the connected GPS (see Section 8.2, GPS Product Protocol Capabilities, on page 43).

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Host → GPS | Pid_Product_Rqst | ignored |
| 1 | Host ← GPS | Pid_Product_Data | Product_Data_Type |

Packet 0 (Pid_Product_Rqst) is a special product request packet that is sent to the GPS. Packet 1 (Pid_Product_Data) is returned to the Host and contains data to identify the GPS, which is provided in the data type 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 GPS product 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 product and are not formatted for display to the end user.

The type definition for the Product_Data_Type is shown below:

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

## 6.2. A001 – Protocol Capability Protocol

The Protocol Capability Protocol is a one-way protocol that allows a GPS to report its protocol capabilities and product-specific data types to the Host. When this protocol is supported by the GPS, it is automatically initiated by the GPS 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 GPS.

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Host ← GPS | 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
    {
    byte                  tag;
    word                  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:

| **tag (byte 0)** | **data (bytes 1 & 2)** | **Notes** |
|---|---|---|
| 'P' | 0 | GPS supports the Default Physical Protocol (P000) |
| 'L' | 1 | GPS supports Link Protocol 1 (L001) |
| 'A' | 10 | GPS supports Device Command Protocol 1 (A010) |
| 'A' | 100 | GPS supports the Waypoint Transfer Protocol (A100) |
| 'D' | 100 | GPS uses Data Type D100 for <D0> during waypoint transfer |
| 'A' | 200 | GPS supports the Route Transfer Protocol (A200) |
| 'D' | 200 | GPS uses Data Type D200 for <D0> during route transfer |
| 'D' | 100 | GPS uses Data Type D100 for <D1> during route transfer |
| 'A' | 300 | GPS supports the Track Log Transfer Protocol (A300) |
| 'D' | 300 | GPS uses Data Type D300 for <D0> during track log transfer |
| 'A' | 500 | GPS supports the Almanac Transfer Protocol (A500) |
| 'D' | 500 | GPS uses Data Type D500 for <D0> during almanac transfer |

The GPS omits the following protocols from the above transmission:

A000 – Product Data Protocol

A001 – Protocol Capability Protocol

A000 is omitted because all products 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 (usually the GPS); for example, the Host might command the GPS to transmit its waypoints. All GPS products are required to implement one of the Device Command protocols, although some commands may not be implemented by the GPS (reception of an unimplemented command causes no error in the GPS; 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 GPS is allowed to initiate a transfer without a command from the other device (for example, when the Host transfers data to the GPS, or when the user presses buttons on the GPS to initiate a transfer).

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → 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 int Command_Id_Type;
```

### 6.3.1. A010 – Device Command Protocol 1

This protocol is implemented by the majority of GPS products (see Section 8.2, GPS Product Protocol Capabilities, on page 43). 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                 */
    };
```

### 6.3.2. A011 – Device Command Protocol 2

This protocol is implemented mainly by panel-mounted aviation GPS products (see Section 8.2, GPS Product Protocol Capabilities, on page 43). 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_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 GPS to send waypoints, the GPS will send every waypoint stored in its database. When the Host sends waypoints to the GPS, the Host may selectively transfer any waypoint it chooses.

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| 1 | Device1 → Device2 | Pid_Wpt_Data | <D0> |
| 2 | Device1 → Device2 | Pid_Wpt_Data | <D0> |
| … | … | … | … |
| n-2 | Device1 → Device2 | Pid_Wpt_Data | <D0> |
| n-1 | Device1 → 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 5.3, Standard Beginning and Ending Packets, on page 12). 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 GPS.

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

## 6.5. A200 – Route Transfer Protocol

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

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| 1 | Device1 → Device2 | Pid_Rte_Hdr | <D0> |
| 2 | Device1 → Device2 | Pid_Rte_Wpt_Data | <D1> |
| 3 | Device1 → Device2 | Pid_Rte_Wpt_Data | <D1> |
| … | … | … | … |
| n-2 | Device1 → Device2 | Pid_Rte_Wpt_Data | <D1> |
| n-1 | Device1 → 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 5.3, Standard Beginning and Ending Packets, on page 12). 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 GPS.

Packet 1 (Pid_Rte_Hdr) contains route header information, which is provided in product-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 product-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.

### 6.5.1. Database Matching for Route Waypoints

Certain products contain an internal database of waypoint information; for example, most aviation products 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 GPS products, the GPS will attempt to match the incoming route waypoints with waypoints in its internal database. First, the GPS inspects the “class” member of the incoming route waypoint; if it indicates a non-user waypoint, then the GPS searches its internal database using values contained in other members of the route waypoint. For aviation units, 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. A300 – Track Log Transfer Protocol

The Track Log Transfer Protocol is used to transfer track logs between devices. Most GPS products store only one track log (called the "active" track log), however, some newer GPS products can store multiple track logs (in addition to the active track log). When the Host commands the GPS to send track logs, the GPS 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 GPS, 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 GPS, the track log is always stored in the active track log within the GPS; i.e., there is no way to transfer track logs into the database of stored track logs. None of these limitations affect GPS products that store only one track log.

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| 1 | Device1 → Device2 | Pid_Trk_Data | <D0> |
| 2 | Device1 → Device2 | Pid_Trk_Data | <D0> |
| … | … | … | … |
| n-2 | Device1 → Device2 | Pid_Trk_Data | <D0> |
| n-1 | Device1 → 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 5.3, Standard Beginning and Ending Packets, on page 12). 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 GPS.

Packets 1 through n-2 (Pid_Trk_Data) each contain data for one track log point, which is provided in product-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.

### 6.6.1. Time Values Ignored by GPS

When the Host transfers a track log to the GPS, the GPS ignores the incoming time value for each track log point and sets the time value to zero in its internal database. If the GPS 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 GPS and track logs that were transferred to the GPS by an external Host.

## 6.7. A400 – Proximity Waypoint Transfer Protocol

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

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| 1 | Device1 → Device2 | Pid_Prx_Wpt_Data | <D0> |
| 2 | Device1 → Device2 | Pid_Prx_Wpt_Data | <D0> |
| … | … | … | … |
| n-2 | Device1 → Device2 | Pid_Prx_Wpt_Data | <D0> |
| n-1 | Device1 → 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 5.3, Standard Beginning and Ending Packets, on page 12). 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 GPS.

Packets 1 through n-2 (Pid_Prx_Wpt_Data) each contain data for one proximity waypoint, which is provided in product-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.

## 6.8. 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 GPS 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 GPS must have current almanac, approximate date and time, and approximate position. Thus, after transferring an almanac to the GPS, the Host should subsequently transfer the date, time, and position (in that order) to the GPS using the following protocols: A600 – Date and Time Initialization Protocol, and A700 – Position Initialization Protocol (see page 23 and 24). After receiving the almanac, the GPS may transmit a request for time and/or a request for position using one of the Device Command protocols.

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

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Records | Records_Type |
| 1 | Device1 → Device2 | Pid_Almanac_Data | <D0> |
| 2 | Device1 → Device2 | Pid_Almanac_Data | <D0> |
| … | … | … | … |
| n-2 | Device1 → Device2 | Pid_Almanac_Data | <D0> |
| n-1 | Device1 → 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 5.3, Standard Beginning and Ending Packets, on page 12). 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 GPS

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

Some product-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.

## 6.9. 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 GPS (see Section 6.8, A500 – Almanac Transfer Protocol, on page 22).

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Date_Time_Data | <D0> |

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

## 6.10. 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 GPS (see Section 6.8, A500 – Almanac Transfer Protocol, on page 22).

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Device1 → Device2 | Pid_Position_Data | <D0> |

Packet 0 (Pid_Position_Data) contains position data, which is provided in product-specific data type <D0>.

## 6.11. A800 – PVT Data Protocol

The PVT Data Protocol is used to provide the Host with real-time position, velocity, and time (PVT) data, which is transmitted by the GPS 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 GPS instead of switching back and forth between NMEA format and GARMIN format.

The Host can turn PVT data on or off by using a Device Command Protocol (see Section 6.3, Device Command Protocols, on page 16). PVT data 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 GPS products turn off PVT data whenever they respond to the Product Data Protocol.

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

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

| N | Direction | Packet ID | Packet Data Type |
|---|---|---|---|
| 0 | Host ← GPS<br>(ACK/NAK optional) | Pid_Pvt_Data | <D0> |

Packet 0 (Pid_Pvt_Data) contains position, velocity, and time data, which is provided in product-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 (such as int, long, float, and double) 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 GPS products use characters from the ASCII character set. Each string type is limited to a specific subset of ASCII characters as shown below:

| | |
|---|---|
| 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 GPS |
| State: | ignored by GPS |
| Facility Name: | ignored by GPS |
| 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 |

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

## 7.3. Basic C Data Types

### 7.3.1. char

The char data type is 8-bit integer or ASCII data. This data type is signed unless the unsigned keyword is present.

### 7.3.2. int

The int data type is 16-bit integer data. This data type is signed unless the unsigned keyword is present.

### 7.3.3. long

The long data type is 32-bit integer data. This data type is signed unless the unsigned keyword is present.

### 7.3.4. float

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

### 7.3.5. double

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

## 7.4. Basic GARMIN Data Types

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

### 7.4.1. 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] = ‘ ’;
```

### 7.4.2. 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
    {
    int                 ABC;
/*  char                XYZ[]           null-terminated string                    */
    int                 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.

### 7.4.3. byte

The byte type is used for 8-bit unsigned integers:

```
typedef unsigned char    byte;
```

### 7.4.4. word

The word type is used for 16-bit unsigned integers:

```
typedef unsigned int     word;
```

### 7.4.5. longword

The longword type is used for 32-bit unsigned integers:

```
typedef unsigned long    longword;
```

### 7.4.6. boolean

The boolean type an 8-bit integer used to indicate true (non-zero) or false (zero):

```
typedef unsigned char    boolean;
```

### 7.4.7. Semicircle_Type

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

```
typedef struct
    {
    long                 lat;          /* latitude in semicircles                        */
    long                 lon;          /* longitude in semicircles                       */
    } Semicircle_Type;
```

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

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

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

### 7.4.8. Radian_Type

The floating point Radian_Type is used to indicate latitude and longitude in radians, where $\pi$ radians equals 180 degrees. North latitudes and East longitudes are indicated with positive numbers; South latitudes and West longitudes are indicated with negative numbers.

```
typedef struct
    {
    double               lat;          /* latitude in radians                            */
    double               lon;          /* longitude in radians                           */
    } Radian_Type;
```

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

degrees = radians * ( 180 / π )

radians = degrees * ( π / 180 )

### 7.4.9. Symbol_Type

The Symbol_Type is used in certain GPS products to indicate the symbol for a waypoint:

```
typedef int Symbol_Type;
```

The enumerated values for Symbol_Type are shown below. Note that most GPS products 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 GPS products. However, the GPS will ignore any unallowed 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 GPS may not accept the requested value.

```
enum
    {
    /*----------------------------------------------------------------
    Symbols for marine (group 0...0-8191...bits 15-13=000).
    ----------------------------------------------------------------*/
    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             */

    /*------------------------------------------------------
    marine navaid symbols
    ------------------------------------------------------*/
    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              */
    /*------------------------------------------------------
    leave space for more navaids (up to 128 total)
    ------------------------------------------------------*/

    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                 */

/*-----------------------------------------------------------------
Symbols for land (group 1...8192-16383...bits 15-13=001).
-----------------------------------------------------------------*/
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                     */

/*-----------------------------------------------------------------
Symbols for aviation (group 2...16383-24575...bits 15-13=010).
-----------------------------------------------------------------*/
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                  */
};
```

## 7.5. Product-Specific Data Types

### 7.5.1. D100_Wpt_Type

Example products: GPS 38, GPS 40, GPS 45, GPS 75 and GPS II.

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

### 7.5.2. D101_Wpt_Type

Example products: GPSMAP 210 and GPSMAP 220 (both prior to version 4.00).

```
typedef struct
    {
    char                ident[6];     /* identifier                         */
    Semicircle_Type     posn;         /* position                           */
    longword            unused;       /* should be set to zero              */
    char                cmnt[40];     /* comment                            */
    float               dst;          /* proximity distance (meters)        */
    byte                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 7.4.9 on page 29). 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 unallowed in the D101_Wpt_Type.

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

### 7.5.3. D102_Wpt_Type

Example products: GPSMAP 175, GPSMAP 210 and GPSMAP 220.

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

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

### 7.5.4. D103_Wpt_Type

Example products: GPS 12, GPS 12 XL, GPS 48 and GPS II Plus.

```
typedef struct
    {
    char                ident[6];     /* identifier                         */
    Semicircle_Type     posn;         /* position                           */
    longword            unused;       /* should be set to zero              */
    char                cmnt[40];     /* comment                            */
    byte                smbl;         /* symbol id                          */
    byte                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_duck           = 12,       /* duck symbol                       */
    smbl_deer           = 13,       /* deer symbol                       */
    smbl_buoy           = 14,       /* buoy 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       */
    };
```

### 7.5.5. D104_Wpt_Type

Example products: GPS III.

```
typedef struct
    {
    char                ident[6];   /* identifier                        */
    Semicircle_Type     posn;       /* position                          */
    longword            unused;     /* should be set to zero             */
    char                cmnt[40];   /* comment                           */
    float               dst;        /* proximity distance (meters)       */
    Symbol_Type         smbl;       /* symbol id                         */
    byte                dspl;       /* display option                    */
    } D104_Wpt_Type;
```

The enumerated values for the “dspl” member of the D104_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 “dst” member is valid only during the Proximity Waypoint Transfer Protocol.

### 7.5.6. D105_Wpt_Type

Example products: StreetPilot (user waypoints).

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

### 7.5.7. D106_Wpt_Type

Example products: StreetPilot (route waypoints).

```
typedef struct
    {
    byte                 class;        /* class                                             */
    byte                 subclass[13]  /* subclass                                          */
    Semicircle_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 “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 GPS map database), the GPS will provide a non-zero value in the “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 GPS (as part of a route), the Host must leave the “class” and “subclass” members unmodified. For user waypoints, the Host must ensure that the “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.

### 7.5.8. D150_Wpt_Type

Example products: GPS 150, GPS 155, GNC 250 and GNC 300.

```
typedef struct
    {
    char                 ident[6];     /* identifier                                        */
    char                 cc[2];        /* country code                                      */
    byte                 class;        /* class                                             */
    Semicircle_Type      posn;         /* position                                          */
    int                  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 “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         */
    };
```

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

### 7.5.9. D151_Wpt_Type

Example products: GPS 55 AVD, GPS 89.

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

The enumerated values for the “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                 */
    };
```

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

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

### 7.5.10. D152_Wpt_Type

Example products: GPS 90, GPS 95 AVD, GPS 95 XL and GPSCOM 190.

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

The enumerated values for the “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                    */
    };
```

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

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

### 7.5.11. D154_Wpt_Type

Example products: GPSMAP 195.

```
typedef struct
    {
    char                 ident[6];      /* identifier                                     */
    Semicircle_Type      posn;          /* position                                       */
    longword             unused;        /* should be set to zero                          */
    char                 cmnt[40];      /* comment                                        */
    float                dst;           /* proximity distance (meters)                    */
    char                 name[30];      /* facility name                                  */
    char                 city[24];      /* city                                           */
    char                 state[2];      /* state                                          */
    int                  alt;           /* altitude (meters)                              */
    char                 cc[2];         /* country code                                   */
    char                 unused2;       /* should be set to zero                          */
    byte                 class;         /* class                                          */
    Symbol_Type          smbl;          /* symbol id                                      */
    } D154_Wpt_Type;
```

The enumerated values for the “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        */
    };
```

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

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

### 7.5.12. D155_Wpt_Type

Example products: GPS III Pilot.

```
typedef struct
    {
    char                 ident[6];     /* identifier                                     */
    Semicircle_Type      posn;         /* position                                       */
    longword             unused;       /* should be set to zero                          */
    char                 cmnt[40];     /* comment                                        */
    float                dst;          /* proximity distance (meters)                    */
    char                 name[30];     /* facility name                                  */
    char                 city[24];     /* city                                           */
    char                 state[2];     /* state                                          */
    int                  alt;          /* altitude (meters)                              */
    char                 cc[2];        /* country code                                   */
    char                 unused2;      /* should be set to zero                          */
    byte                 class;        /* class                                          */
    Symbol_Type          smbl;         /* symbol id                                      */
    byte                 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 “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                    */
    };
```

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

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

### 7.5.13. D200_Rte_Hdr_Type

Example products: GPS 55 and GPS 55 AVD.

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

### 7.5.14. D201_Rte_Hdr_Type

Example products: all products unless otherwise noted.

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

### 7.5.15. D202_Rte_Hdr_Type

Example products: StreetPilot.

```
typedef struct
    {
/*  char                 rte_ident[];     null-terminated string                                  */
    } D202_Rte_Hdr_Type;
```

### 7.5.16. D300_Trk_Point_Type

Example products: all products unless otherwise noted.

```
typedef struct
    {
    Semicircle_Type      posn;          /* position                                               */
    longword             time;          /* time                                                   */
    boolean              new_trk;       /* new track segment?                                     */
    } D300_Trk_Point_Type;
```

The “time” member provides a timestamp for the track log point. This time is expressed as the number of seconds since 12:00 AM on January 1st, 1990.

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

### 7.5.17. D400_Prx_Wpt_Type

Example products: GPS 55 and GPS 75.

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

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

### 7.5.18. D403_Prx_Wpt_Type

Example products: GPS 12, GPS 12 XL and GPS 48.

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

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

### 7.5.19. D450_Prx_Wpt_Type

Example products: GPS 150, GPS 155, GNC 250 and GNC 300.

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

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

### 7.5.20. D500_Almanac_Type

Example products: GPS 38, GPS 40, GPS 45, GPS 55, GPS 75, GPS 95 and GPS II.

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

### 7.5.21. D501_Almanac_Type

Example products: GPS 12, GPS 12 XL, GPS 48, GPS II Plus and GPS III.

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

### 7.5.22. D550_Almanac_Type

Example products: GPS 150, GPS 155, GNC 250 and GNC 300.

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

### 7.5.23. D551_Almanac_Type

Example products: GPS 150 XL, GPS 155 XL, GNC 250 XL and GNC 300 XL.

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

### 7.5.24. D600_Date_Time_Type

Example products: all products unless otherwise noted.

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

The D600_Date_Time_Type contains the UTC date and UTC time.

### 7.5.25. D700_Position_Type

Example products: all products unless otherwise noted.

```
typedef Radian_Type D700_Position_Type;
```

### 7.5.26. D800_Pvt_Data_Type

Example products: GPS III and StreetPilot.

```
typedef struct
   {
   float             alt;         /* altitude above WGS 84 ellipsoid (meters)      */
   float             epe;         /* estimated position error, 2 sigma (meters)    */
   float             eph;         /* epe, but horizontal only (meters)             */
   float             epv;         /* epe, but vertical only (meters)               */
   int               fix;         /* type of position fix                          */
   double            tow;         /* time of week (seconds)                        */
   Radian_Type       posn;        /* latitude and longitude (radians)              */
   float             east;        /* velocity east  (meters/second)                */
   float             north;       /* velocity north (meters/second)                */
   float             up;          /* velocity up    (meters/second)                */
   float             msl_hght;    /* height of WGS 84 ellipsoid above MSL (meters) */
   int               leap_scnds;  /* difference between GPS and UTC (seconds)      */
   long              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 January 1st, 1990 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 January 1st, 1990 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 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 GPS 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                */
    };
```

# 8. Appendixes

## 8.1. GPS Product IDs

The table below provides the Product ID numbers for many GARMIN GPS products.

| 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 55 | 14 |
| GPS 55 AVD | 15 |
| GPS 65 | 18 |
| GPS 75 | 13 |
| GPS 75 | 23 |
| GPS 75 | 42 |
| GPS 89 | 39 |
| GPS 90 | 45 |
| 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 220 | 29 |
| GPSMAP 230 | 49 |
| GPSMAP 230 Chinese | 76 |
| GPSMAP 235 Sounder | 49 |

## 8.2. GPS Product Protocol Capabilities

The table below provides the protocol capabilities of many GARMIN GPS products that do not implement the Protocol Capabilities Protocol. Column 1 contains the applicable Product ID number, and Column 2 contains the applicable software version number. The remaining columns show the product-specific protocol IDs and data type

IDs for the types of protocols indicated (Wpt, Rte, Trk, Alm, Prx, and PVT). 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 product in the table below indicates that the product did not originally implement the Protocol Capabilities Protocol (A001). However, if the Host detects that one of these products now provides Protocol Capabilities Protocol data (due to a new version of software loaded in the product), 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 products in the table implement them:

P000 Default Physical Protocol
A000 Product Data Protocol
A600 Date and Time Initialization Protocol
A700 Position Initialization Protocol

All products in the table use the D600 data type in conjunction with the A600 protocol; similarly, all products 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 products in the table implements PVT Data transfer.

Note: all numbers are in decimal format.

| ID | Version | Link | Cmnd | Wpt | Rte | Trk | Prx | Alm |
|---|---|---|---|---|---|---|---|---|
| 13 | All | L001 | A010 | A100, D100 | A200, D201, D100 | A300, D300 | A400, D400 | A500, D500 |
| 14 | All | L001 | A010 | A100, D100 | A200, D200, D100 | A300, D300 | A400, D400 | A500, D500 |
| 15 | All | L001 | A010 | A100, D151 | A200, D200, D151 | A300, D300 | A400, D151 | A500, D500 |
| 18 | All | L001 | A010 | A100, D100 | A200, D201, 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, D201, D152 | A300, D300 | A400, D152 | A500, D500 |
| 23 | All | L001 | A010 | A100, D100 | A200, D201, D100 | A300, D300 | A400, D400 | A500, D500 |
| 24 | All | L001 | A010 | A100, D100 | A200, D201, 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, D201, D100 | A300, D300 | A400, D400 | A500, D500 |
| 36 | < 3.00 | L001 | A010 | A100, D152 | A200, D201, D152 | A300, D300 | A400, D152 | A500, D500 |
| 36 | >= 3.00 | L001 | A010 | A100, D152 | A200, D201, 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, D201, 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 |
| 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 |
| 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 |

## 8.3. Frequently Asked Questions

### 8.3.1. Undocumented Protocols

Q: The Internet has information about additional protocols and extensions that are not described in the document. Why have these been left out?

A: Part of the goal of the document is to separate what GARMIN thinks is safe versus what is unsafe when interfacing to our GPS products. Any items left out of the document are considered to be “testing aids” for use by our engineering and manufacturing departments only. As such, we do not require all products to have all testing aids, nor do we require the testing aids to be implemented in the same way in every product. In fact, there is a wide variation in these testing aids. Worse, some testing aids may have side effects that are undesirable for anything but testing.

### 8.3.2. 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.

#### 8.3.3. 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 GPS?

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 or the lookup table in Section 8.2 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 GPS 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).

#### 8.3.4. Waypoint Creation Date

Q: Isn't the “unused” longword in waypoint formats really the date of waypoint creation?

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

#### 8.3.5. 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.

#### 8.3.6. Example Code

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

A: We are currently unable to take the time to compile this information.

#### 8.3.7. Sample Data Transfer Dumps

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

A: We are currently unable to take the time to compile this information.

### 8.3.8. 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.

### 8.3.9. Software Versions

Q: Why doesn't the table in Section 8.1 include an indication of software version?

A: We are currently unable to take the time to compile this information. The purpose of the table is to allow you to determine the Product IDs for the products 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 the table in Section 8.2.