HARRIS CORPORATION

Crucial Security, Inc.

Per-Packet Information specification for Geolocation

Jon “Johnny Cache” Ellch, Senior Security Researcher
jellch@harris.com

Crucial Security, Inc.
14900 Conference Center Drive
Suite 225
Chantilly, VA 20151

VERSION 2.0.0-d21

July 12, 2011

TABLE OF CONTENTS

Paragraph **Title** **Page**

0. Table of Contents

1. Introduction

When capturing live networking data, it is often useful to store out-of-band meta information alongside the actual packet contents. Historically this has been most useful for wireless networks, where information such as signal strength and channel number are of particular importance.

There have been a number of ad-hoc solutions to this problem in the past (prism headers, AVS headers, and more recently radiotap headers). None of these provided a general solution for allowing arbitrary meta-information to be stored inline with a packet however.

CACE technologies has developed a general solution to this problem known as *Per-Packet Information* (PPI). Tools that implement PPI allow arbitrary data to be included with individual packets in a standard libpcap formatted packet capture. The PPI specification can be downloaded from CACE at http://www.cacetech.com/documents/

1.1. *Purpose*

The purpose of this specification is to document a series of PPI tags that can be used to describe the context in which an 802.11 (or any other) packet was captured. By utilizing a common format it is the hope of the author that a level of interoperability will be achieved across different tools. This will enable the de-coupling of packet visualization and analysis from the platform they are collected on. Four unique tags are described in this paper, and are collectively referred to as the GEOLOCATION-TAGS.

PPI GEOLOCATION-TAGS		
Number	**Name**	**Description**
30002	GPS	An extendible tag that can carry Lon/Lat/Alt as well as error levels and a 64-bit timestamp. Despite the name, it may be populated by non-GPS devices (INS, software, etc).
30003	VECTOR	An extendible tag that carries all rotational and offset information. Data encoded in a VECTOR field may come from different sources (GPS's and digital compass's being the most common). VECTOR-TAGs can be used to represent many things, the most useful being antenna/vehicle orientation.
30004	SENSOR	Used to store arbitrary sensor data that is associated with the current reference frame. Most commonly used for velocity and acceleration.
30005	ANTENNA	An extendible tag that can carry information about the antenna utilized to collect the packet. Properties include horizontal and vertical beamwidth, as well as gain.

Another tag, already defined and documented by CACE, is also explicitly recognized by the PPI-GEOLOCATION standard. This tag, 802.11-COMMON, is defined in the PPI specification. Its description here is only for informational purposes.

2	802.11-	Loosely based off radiotap standard, can be used to

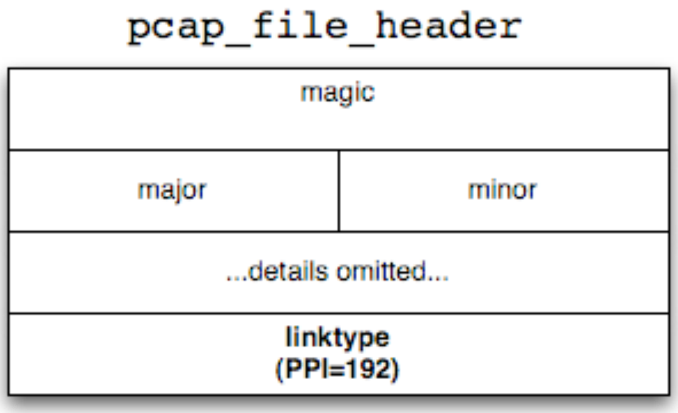
	COMMON	by advanced PPI-GEOLOCATION applications to indicate signal strength on a per-interface or per-antenna (vs per-packet) level.

2. Format

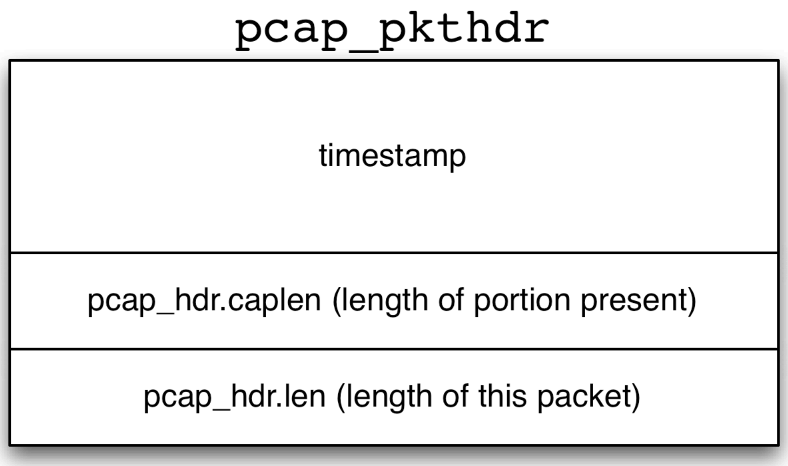
The following section describes the layout on disk of a pcap file that contains PPI encapsulated data. The description of the pcap and PPI headers are for informational purposes only; they are defined outside the scope of this document. Detailed definitions of the GEOLOCATION-TAGS are presented in sections 3 through 6.

2.1. *Format overview*

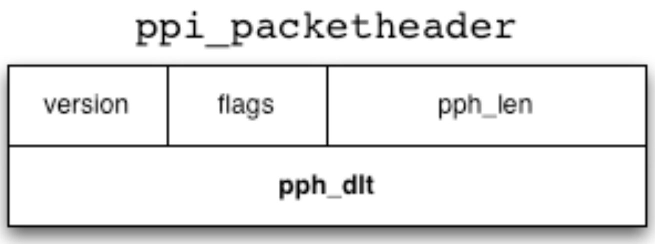
Every pcap file begins with a pcap_file_header. This fixed size header contains a magic number, version information, and a data linktype (often abbreviated DLT). The DLT field specifies the type of packets encapsulated in the file. Commonly used DLTs include Ethernet (DLT_EN10MB) and 802.11 (DLT_IEEE802_11). When PPI is in use the DLT will be set to DLT_PPI (192).



Immediately following the pcap_file_header, a pcap_pkthdr will be found. Every packet encapsulated in a pcap file has a pcap_pkthdr. This header contains a timestamp, the total length of the packet, and the number of bytes actually captured (may be less than the total). Both length fields in a pcap_pkthdr include the PPI data that follows.

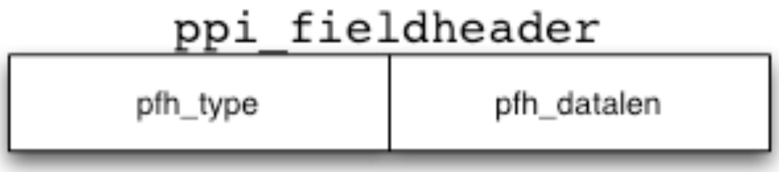

pcap_pkthdr
timestamp
pcap_hdr.caplen (length of portion present)
pcap_hdr.len (length of this packet)

Immediately following the pcap_pkthdr, the linktype specific information begins. In the case of DLT_PPI the next header encountered is a ppi_packetheader.



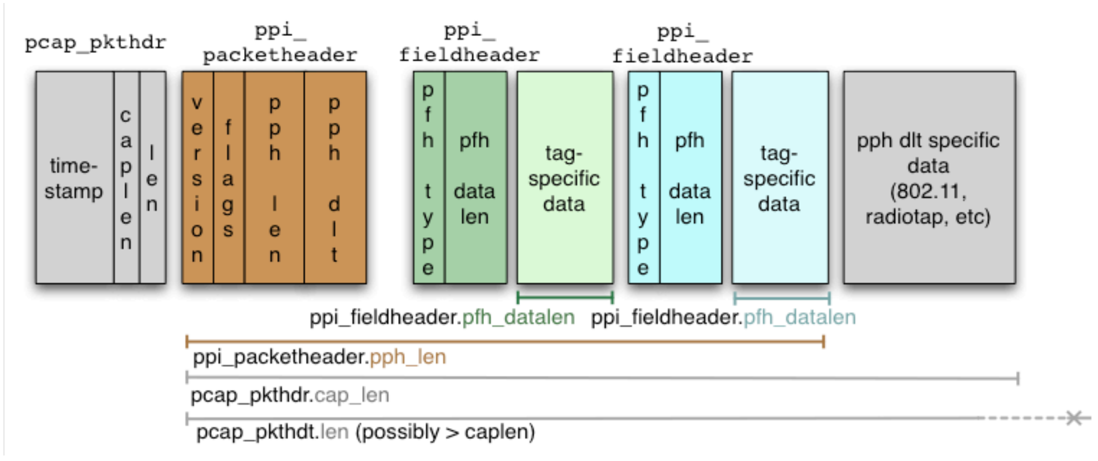
This header specifies the version, length, and data link type of the following packet. The pph_len field accounts only for PPI meta-information PPI encapsulated data. It does not include data farther up the stack (for example, 802.11 headers).

After the ppi_packetheader, zero or more ppi_fieldheaders will be found. There is one ppi_fieldheader for each PPI tag included in the file.

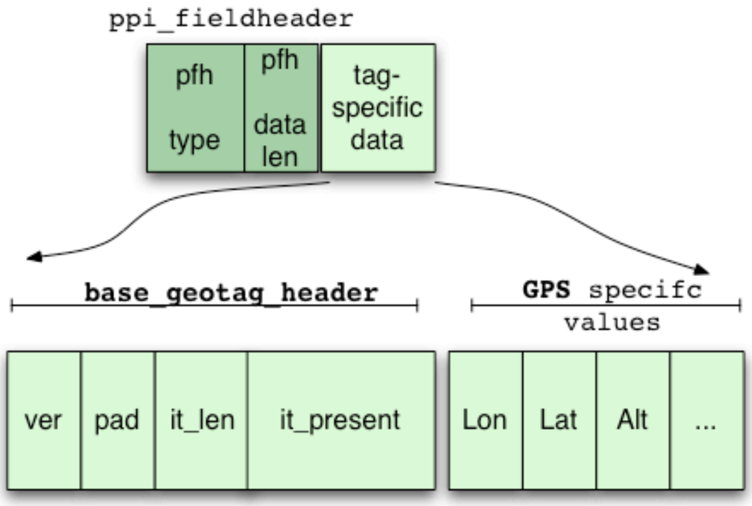


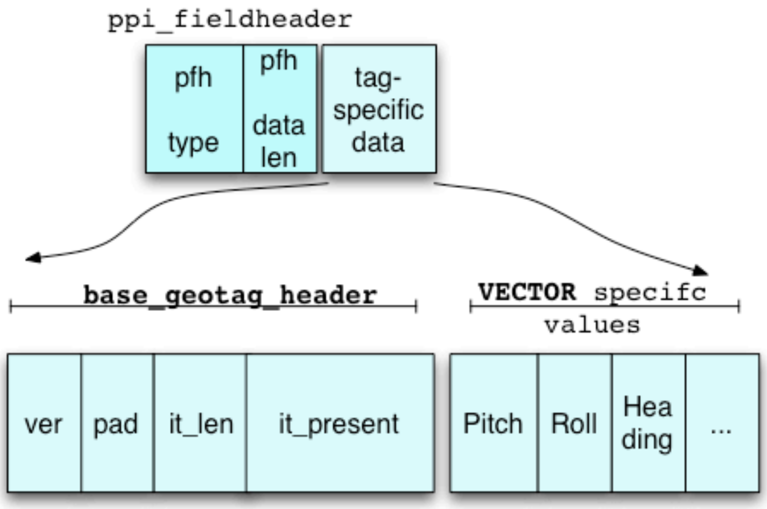
The pfh_type field is a 16-bit tag that specifies the type of data that follows (30002 for GPS information, for example). The pfh_datalen field specifies the length of the data that follows (must be between 0 and 65,520 inclusive). Following each ppi_fieldheader is a block of tag-specific pfh_datalen data.

The diagram below shows a pcap file where the first packet has two unique ppi_fieldheaders.



These two ppi_fieldheaders could correspond to a GPS and VECTOR tag. Assuming that is the case, we would have a packet that looked like the following diagram.





2.2. Format details

This section describes the GEOLOCATION-TAGS in detail. Although the GEOLOCATION-TAGS are logically independent, they share many implementation details. In particular, they share a capability for specifying the presence of various fields, the entire length of a particular tag, and a cross platform way to efficiently represent fixed point numbers. Before describing the GEOLOCATION-TAGS a brief note on radiotap headers is in order.

GEOLOCATION-TAGS have been influenced heavily by the format of radiotap headers. Radiotap headers are the latest in a series of ad-hoc solutions to encoding important characteristics about 802.11 packets in-line with packet captures. For example, radiotap headers can specify the channel and received signal strength that a packet was captured with, as well as many other characteristics.

PPI-GEOLOCATION tags are designed to be particularly easy for developers familiar with radiotap to read and write. Readers unfamiliar with radiotap may wish to familiarize themselves with it before implementing any PPI-GEOLOCATION parsers.

2.2.1. Basic geotagging header

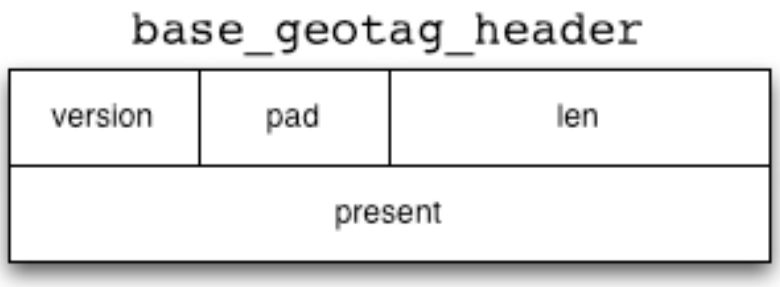
All GEOLOCATION-TAGS begin with a `base_geotag_header`. This header is intentionally bit-for-bit compatible with a radiotap header. This design decision allows commonly available radiotap parsers to be re-purposed by initializing them with different tables for the field sizes and descriptions. The base tag header (logically equivalent to an `ieee80211_radiotap_header`) is described below.

```
Struct base_geotag_header {
u_int8_t        version;     //set to 1
u_int8_t        pad;
u_int16_t       len;         //entire length
u_int32_t       present;     //bitmask indicating fields present
} __attribute__((__packed__));
```

The version field will change only for updates that would break previous parsers. Currently all GEOLOCATION-TAGS set this to **2**. Version 1 was circulated under 1.2.0. Version 1 did not define sensor tags, and had acceleration/velocity stored in Vector tags. Version 0 was utilized for internal development purposes, and should not be used in the wild. *Version 2 is the first widely implemented version of this specification.*

The pad field serves only to make the len field naturally aligned.

The len field specifies the length of the current individual tag *including* the `base_geotag_header`. This value must be between 8 and 65,535. (The `base_geotag_header.len` field is redundant with the `ppi_fieldheader.data_len` field. Its inclusion is convenient for standalone parsers, which may be unaware of the PPI encapsulation.)



The 32-bit present field is a bitmask specifying which fields are present in this GPS, VECTOR, SENSOR, or ANTENNA tag. The use of the bitmask scheme allows for tags to contain a subset of all possible data (for example, longitude and latitude, but not altitude). The tag-specific bit definitions are covered in sections 3 through 6.

The following properties are true of both radiotap and `base_geotag_header` parsing:

- Fields are strictly ordered; The developer can specify any combination of fields, but the data must appear in the same order as the set bit numbers in the `present` bitmask.
- All data fields including the `version`, `len`, and `present` fields in the `base_geotag_header` are to be specified in little endian byte-order.
- Field lengths are implicit: the header format does not specify field lengths, it is expected that the developer knows the corresponding length based on the data field name.
- Variable-length fields are not supported since field lengths are implicit.
- The MSB of the `present` bitmask is utilized to indicate the presence of an extended bitmask following it, although this is currently unused.

While the list of similarities is extensive, there is one significant difference between radiotap headers and base`_geotag_headers`: `Base_geotag_headers` **do *not* force natural alignment.**

The requirement for natural alignment in radiotap was (assumedly) made because time-sensitive kernel level code is involved in writing radiotap headers. It is expected that GEOLOCATION-TAGS will be handled in userland, and therefore the need for natural alignment is less compelling.

2.2.2. Fixed point numerical representation

GEOLOCATION-TAGS need to represent a diverse range of floating point numbers on disk in a platform neutral manner. The details on how floating point values are stored on disk are covered in this section.

A majority of floating point values processed by GEOLOCATION-TAGS can be handled as fixed point values ranging between 000.000000 and 999.999999. These values are said to be `fixed3_6` encoded (the 3_6 denoting the number of digits on either side of the decimal point).

Values stored in the `fixed3_6` format include GPS error margins, compass bearings, and others. A reference encoding/decoding function is presented in section 2.2.2.1, as well as a table illustrating the encoding scheme.

While the `fixed3_6` encoding is sufficient for expressing many values, some fields require more specialized formats. These fields either need to represent negative as well as positive values, or have unique requirements on the location of the decimal point. In particular, longitude, latitude and altitude require special cases to ensure an optimal range and precisions are utilized. These specialized fixed-point formats are detailed in Sections 2.2.2.1, 2.2.2.2, and 2.2.2.3.

The following table summarizes the features of each encoding format.

Standard GEOLOCATION-TAG Number Encoding Formats

Name	Section	Range	Precision	Use
`fixed3_6`	2.2.2.1	`000.000000` `+999.999999`	3.6	Position error estimates, Angular rotations and error estimates, Antenna Beamwidth and Gain.
`Fixed3_7`	2.2.2.2	`-180.0000001` `+180.0000001`	3.7	Latitude and Longitude.
`Fixed6_4`	2.2.2.3	`-180000.0001` `+180000.0001`	6.4	Altitude, Position offsets, Velocity and Acceleration

2.2.2.1. Fixed 3_6 encoding

```
//Input: a positive floating point value
//between 000.0000000 and 999.9999999
//Output: a LITTLE ENDIAN (not necessarily native)
//32 bit unsigned value between 0 and 999999999
//Returns: 0 on success
//          -1 on input value to negative
//          -2 on input value to positive
int flt_to_fixed3_6(double l, u_int32_t &out)
{
    if(l <= -000.000001)
    {
      fprintf(stderr, “flt_to_fixed3_6: Error. Input value too neg to
convert. %f\n”,l);
      return -1;

    }
    if(l > +999.999999)
    {
      fprintf(stderr, “flt_to_fixed3_6: Error. Input value too pos to
convert. %f\n”,l);
      return -2;
    }
    out = l * 1000000;
    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(out);
    #endif
    return 0;
}
```

Table 1: `fixed3_6` encoding examples

Value:	Encoded (decimal)	Encoded: (little endian)	Comment
000.000000	0000000000	0x00000000	Zero
+000.000001	0000000001	0x01000000	One millionth
+001.000000	0001000000	0x42420f00	One
+123.123456	0123123456	0x00b75607	
+360.000000	0360000000	0x002A7515	
+999.999999	0999999999	0xFFC99A3B	Largest legal value
+1000.000000	1000000000	0x00CA9A3B	First illegal value

```
//Input: a LITTLE ENDIAN (not necessarily native)
//32 bit unsigned value between 0 and 999999999
//Output: a positive floating point value
//between 000.0000000 and 999.9999999
//Returns: 0 on success
//        -2 on input value to positive
int fixed3_6_to_flt(u_int32_t l, double &out)
{
    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(l);
    #endif
    if(l >= +1000000000)
    {
      fprintf(stderr, "flt_to_fixed3_6: Error. Input value too pos to
convert. %f\n",l);
      return -2;
    }
    out = (double) l / 1000000.0;
    return 0;
```

2.2.2.2. fixed_3_7 encoding (Longitude/Latitude)

Most programs represent longitude and latitude as a floating point value between -180.0000000 and +180.0000000. In order to eliminate the need to store a sign, these values are mapped to the range (0, 3600000000), with a fixed decimal point after the third digit. This allows any longitude/latitude to be represented in 4 bytes with 7 digits to the right of the decimal. Any values between 3600000001 (0x0xD693A401) and 4294967295 (0xFFFFFFFF) are invalid. A sample function that performs this mapping is shown below.

Table 2: `fixed3_7` encoding samples

Value:	Encoded (decimal)	Encoded: (little endian)	Comment
-180.0000001			Illegal value
-180.0000000	0000000000	0x00000000	Smallest legal value
-179.9999999	0000000001	0x01000000	
000.0000000	1800000000	0x00D2496B	Zero
+123.1234567	3031234567	0x07F8ACB4	
+179.9999999	3599999999	0xFFA393D6	
+180.0000000	3600000000	0x00A493D6	Largest legal value
+180.0000001	3600000001	0x01A493D6	Illegal value

```
//Input: a signed floating point value (latitude/longitude are good examples)
//between -180.0000000 and +180.0000000, inclusive)
//Output: a LITTLE ENDIAN (not necessarily native)
//32 bit unsigned value between 0 and 3600000000
//Returns: 0 on success
//         -1 on input value to negative
//         -2 on input value to positive
int flt_to_fixed_3_7(double l, u_int32_t &out)
{
    if(l <= -180.0000001)
    {
      fprintf(stderr, “ppi_gpstag_encode_lon_lat: Error. Input value too neg
      to convert. %f\n”,l);
      return -1;
    }

    if(l >= +180.0000001)
    {
      fprintf(stderr, “ppi_gpstag_encode_lon_lat: Error. Input value too pos
      to convert. %f\n”,l);
      return -2;
    }
    //scaled_l may be positive or negative.
    Int32_t scaled_l =  (int32_t) ((l) * (double) 10000000);
    out = (u_int32_t) (scaled_l + ((int32_t) 180 * 10000000));
    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(out);
    #endif
    return 0;
}
```

```
//Input: a LITTLE ENDIAN (not necessarily native) 32 bit unsigned
//value between 0 and 3600000000
//Output: a signed floating point (suitable for latitude or longitude)
//between -180.0000000 and +180.0000000, inclusive)
//Returns: 0 on success
//         -2 on input value to positive
int fixed_3_7_to_flt(u_int32_t l, double &out)
{
    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(l);
    #endif
    if ( l > 3600000000)
     {
      fprintf(stderr, "ppi_gpstag_decode_lon_lat: Error.
                       Input value too pos to convert. %f\n",l);
      return -2;
    }
    int32_t remapped_l = l – (180 * 10000000);
    out = (double) ((double) remapped_l / 10000000);
    return 0;
}
```

2.2.2.3. fixed_6_4 encoding (Altitude, Sensor Data)

Some floating point values, such as altitude, need to express a larger range with less precision. The fixed_6_4 encoding scheme can be used in these cases.

For example, most altitudes range between -10,000m (Pacific Ocean's Marianas Trench) and +10,000m (commercial aircraft). By utilizing the following encoding scheme (similar to fixed_3_6) we can map (-180000.0000, +180000.0000) onto (0, 3600000000). This provides sufficient range to express altitudes on Earth with millimeter level precision.

This format is also used for all Sensor data in conjunction with a scaling factor term, which allows a general signed, floating point format with 9+ significant digits over the range of 10^{-128} to 10^{127}. See Section 5 for details on Sensor Tags and Sensor Data types.

Table 3: fixed_6_4 encoding samples

Value:	Encoded (decimal)	Encoded (little endian)	Comment
-180000.0001			Illegal value
-180000.0000	0000000000	0x00000000	Most negative expressible value
-179999.9999	0000000001	0x01000000	
-010000.0000	0800000000	0x0008af2f	Marianas trench (approx)
000000.0000	1800000000	0x00D2496B	Sea level
+000000.0001	1800000001	0x01D2496B	Sea level plus .0001 meters
+021000.0123	2010000123	0xfb2ace77	Very high altitude flight
+179999.9999	3599999999	0xFFA393D6	
+180000.0000	3600000000	0x00A493D6	Most positive expressible value
+180000.0001			Illegal value

```
//Input: a signed floating point value (f.ex, altitude) between
//-180000.0000 and +180000.0000 meters, inclusive)
//Output: a LITTLE ENDIAN (not necessarily native)
//        32 bit unsigned value between 0 and 3600000000
//Returns: 0 on success
//         -1 on input value to negative
//         -2 on input value to positive
int flt_to_fixed_6_4(double l, u_int32_t &out)
{
    if(l <= -180000.0001)
    {
      fprintf(stderr, “ppi_gpstag_encode_lon_lat: Error. Input value too neg
      to convert. %f\n”,l);
      return -1;
    }

    if(l >= +180000.0001)
    {
      fprintf(stderr, “ppi_gpstag_encode_lon_lat: Error. Input value too pos
      to convert. %f\n”,l);
      return -2;
    }
    //scaled_l may be positive or negative.
    Int32_t scaled_l =  (int32_t) ((l) * (double) 10000);
    out=(u_int32_t) (scaled_l + ((int32_t) 180000 * 10000));
    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(out);
    #endif
    return 0;
}
```

```
//Input: a LITTLE ENDIAN (not necessarily native)
//32 bit unsigned value between 0 and 3600000000
//Output: a signed floating point value (f.ex, altitude)
//between -180000.0000 and +180000.0000 meters, inclusive)
//Returns: 0 on success
//         -2 on input value to positive
int fixed_6_4_to_flt(u_int32_t alt, double &out)
{

    #if BYTE_ORDER == BIG_ENDIAN
    SWAP4(alt);
    #endif
    if(alt >= +3600000001)
    {
      fprintf(stderr, “ppi_gpstag_decodea_alt: Error. Input value too pos to
      convert. %f\n”,alt);
      return -2;
    }
    int32_t remapped_alt = alt – (180000 * 10000);
    out = (double) ((double) remapped_alt / 10000);
    return 0;
}
```

2.2.3. Encoding Application Specific data

As currently defined, each PPI-GEOLOCATION tag includes an optional, 4-byte application identifier (AppId) and an optional application-specific 60-byte field (AppData). These fields can be used to identify which application generated a GEOLOCATION-TAG, as well as store small amounts of application-specific data. They are indicated by the presence of bits 29 and 30 in the present bitmask.

2.2.3.1. Application ID

The Application ID (AppId) is a 4-byte field that can be used to identify the application that generated the current GEOLOCATION-TAG. Applications that utilize advanced feature of the GEOLOCATION-TAGS specification are encouraged to include an AppId in every GEOLOCATION-TAG to aid in diagnosing errors in the specification.

Application ID numbers are intended to be self-regulating. Current defined values for AppId are included in the table below. All other values are unallocated. Applications may choose any unallocated app number they wish. Developers are encouraged to contact the maintainer of this standard to have their application number published.

Value (on disk: little endian)	Description
00 00 00 00 – 00 00 00 FF	Reserved
01 02 03 04	Reserved, testing
53 52 48 00 – 53 52 48 FF	Harris, Inc (HRS)
53 49 4B 00 – 53 49 4B FF	Kismet development (KIS)
4D 4F 52 00 – 4D 4F 52 FF	Reserved
4A 46 43 00 – 4A 46 43 FF	Reserved (JFC)
52 41 47 00 – 52 41 47 FF	Reserved (RAG)
53 52 54 00 – 53 52 54 FF	Reserved (SRT)
31 33 70 00 – 31 33 70 FF	Reserved
FF 00 00 00 – FF 00 00 FF	Reserved

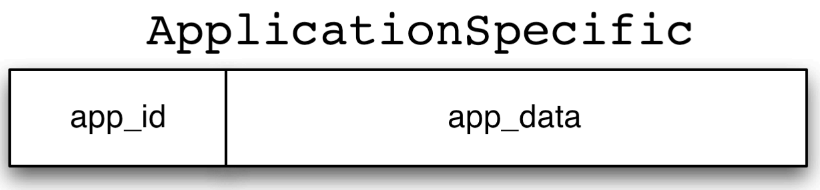
2.2.3.2. Application Data

All GEOLOCATION-TAGS include the ability to carry 60 bytes of arbitrary data in the AppData field. Developers who need to store small amounts of application specific data related to the tag in question can use this field. The AppData field is indicated by bit 30 in the present bitmask.

Applications that utilize the 60-byte AppData field **must** also include an AppId field (in order to correctly identify the format of the AppData field). Parsing and versioning of application specific data is up to the individual application.

When the AppData field is utilized, it can be logically considered as a 64-byte field of the following format (since the app_num must be present as well).

```
Struct app_specific{
u_int32_t       app_id;     //32-bit little-endian app number.
U_int8_t        app_data[60]; //application specific
} __attribute__((__packed__));
```



Developers utilizing the AppData field are encouraged to document their formats, and to request a proper field in the PPI-GEOLOCATION TAG if applicable. Developers may also consider creating their own SENSOR type tag if applicable.

3. GPS-TAG

The GPS-TAG is used to provide a general frame of reference for the tagging of a packet. The GPS-TAG can encode lon/lat/alt, error margins, and a high-resolution timestamp. It can also encode what device type was used (GPS, INS, etc).

Advanced applications that need to encode data captured with higher resolution than offered by the GPS-TAG (for example, 2 antennas a fixed width apart) can accomplish this via the use of offset fields in the VECTOR-TAG (covered in section 4). Simpler applications can simply tag each packet with a lon/lat/alt by using the GPS-TAG itself.

Like all GEOLOCATION-TAGS, the GPS tag begins with a `base_geotag_header`. The following table describes the bitmask to be used with this tag:

Bit (LSB=0)	**Length**	**Name**	**Encoding** (unit, format)
0	4	GpsFlags	32-bit bitmask
1	4	Latitude	degrees, `fixed3_7`
2	4	Longitude	degrees, `fixed3_7`
3	4	Altitude	meters, `fixed6_4`
4	4	Altitude_g	meters, `fixed6_4`
5	4	Gpstime:	Details below
6	4	FractionalTime	ns, 32-bit unsigned
7	4	eph:	meters, `fixed3_6`
8	4	epv	meters, `fixed3_6`
9	4	ept:	ns, 32-bit unsigned
10-27	0	Reserved	
28	32	DescriptionString	ASCII, null-padded
29	4	AppId	See section 2.2.3.1
30	60	AppData	See section 2.2.3.2
31	0	Indicates extended bitmap.	

Immediately following the `base_geotag_header` zero or more fields (as specified in the present field) will follow. They shall be ordered by the significance of the bit used to specify the field in the present bitmask (increasing), and encoded according to the table above.

As currently defined a valid GPS-TAG shall not be any larger than **144** bytes (the size of a `base_geotag_header` plus every currently defined field), or less than 8 bytes (the size of a `base_geotag_header`). Any GEOLOCATION-TAG that does not include at least a `base_geotag_header` is invalid.

3.1. GpsFlags bitmask

The GpsFlags bitmask is used to describe the type of positioning system that was used to provide a location. A likely source of its values is the NMEA GPGGA field "Quality Indicator" field.

Bit	Description
0	No fix available
1	GPS fix
2	Differential GPS fix
3	PPS fix
4	Real Time Kinematic
5	Float RTK
6	estimated (dead reckoning)
7	Manual input mode
8-31	Reserved

3.2. GPS-TAG fields

Name	**Description**
Latitude	Self-descriptive, `fixed3_7` encoding
Longitude	Self-descriptive, `fixed3_7` encoding
Altitude	Self-descriptive, `fixed6_4` encoding
Altitude_g	Altitude **from ground level**, `fixed6_4` encoding. See 9.5.1 for processing details.
Gpstime	32-bit unsigned counter, seconds since unix epoch (UTC). Like all Fields contained in GEOLOCATION-TAGS, this is stored little endian.
Fractional Time	32-bit unsigned counter, 1 nano-second resolution. Fractional time should not exceed one second.
eph	Estimated horizontal error in meters. Stored as a `fixed3_6`
epv	Estimated vertical error in meters. Stored as a `fixed3_6`
ept	Estimated clock error in nano-seconds. Stored as a 32-bit unsigned counter. (Utilizing ns as units allows between 0 and 4 secs of clock error, while maximizing resolution).
DescriptionString	32-byte NULL padded ASCII description of what this GPS-TAG is attached to. Examples include "Silver ford Taurus" or "Stationary-antenna-1"
AppId	See section 2.2.3.1
AppData	See section 2.2.3.2

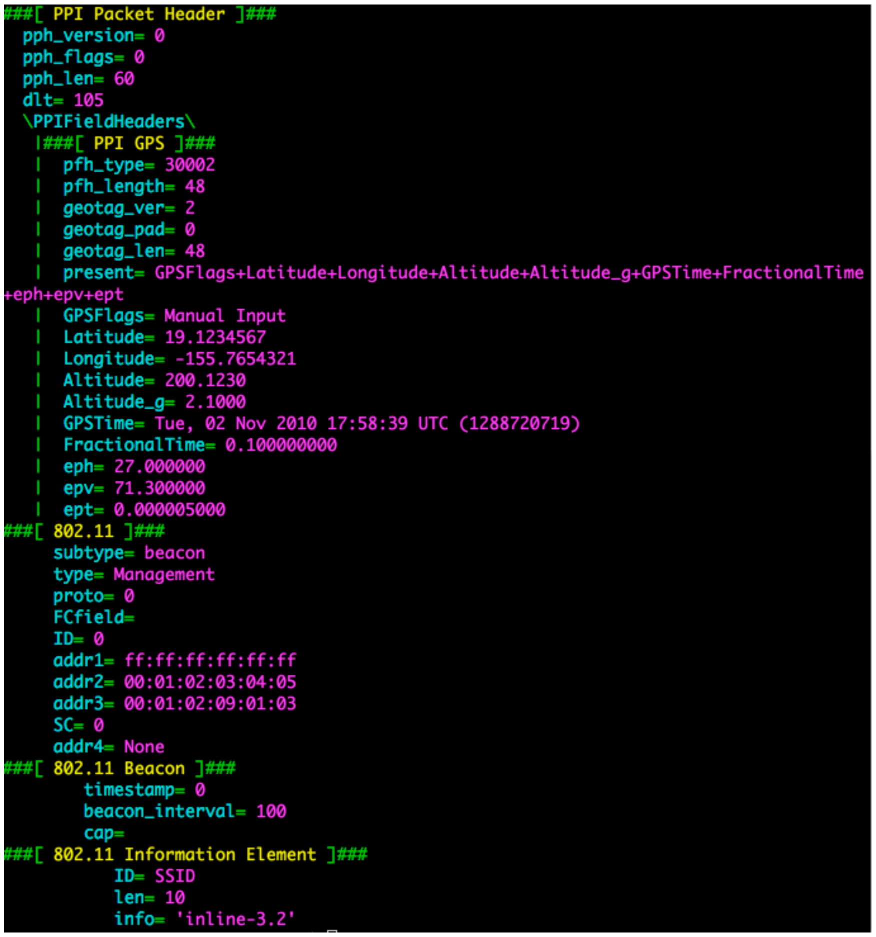
Maximum GPS-TAG size:

Producers of error values (eph, epv, ept) are expected to store them with at least 95% confidence. If an application cannot determine the error within a 95% range the value should not be stored.

Applications consuming eph, epv, or ept may assume that they were stored with at least 95% confidence.

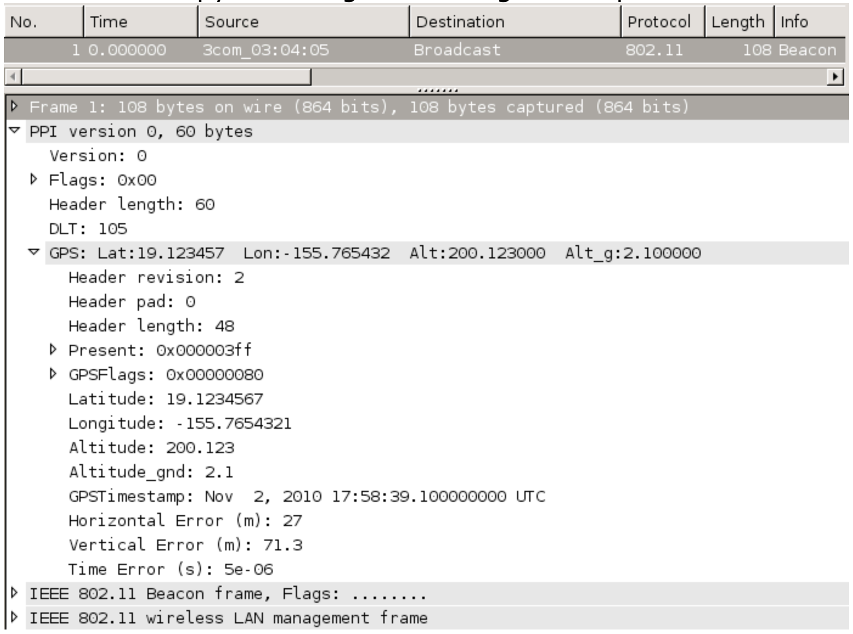
A concrete example of the GPS-TAG encoding scheme is shown below. Note that the inclusion of Altitude and Altitude_g is for illustration purposes only. Real life applications are expected to utilize only one representation of altitude.

GPS-TAG Example			
Field	**Native**	**Decimal**	**Encoded** (little endian)
GPSFlags	Manual fix		0x80000000
Latitiude	19° 7’24.45”N	19.1234567	0x07D4AF76
Longitude	155°36’54.23”W	-155.7654321	0xCFE6710E
Altitude		200.123m	0x4E5B686B
Altitude_g		002.100m	0x08244A6B
Date	Tue, 02 Nov 2010 17:58:39 UTC	1288720719	0x4F51D04C
FractionalTime	0.1s	0.1	0x00E1F505
Horiz-err		27.0m	0xC0FC9B01
Vert-err		71.3m	0xA0F33F04
Time-err		.000005s	0x88130000



```
###[ PPI Packet Header ]###
  pph_version= 0
  pph_flags= 0
  pph_len= 60
  dlt= 105
  \PPIFieldHeaders\
   |###[ PPI GPS ]###
   |  pfh_type= 30002
   |  pfh_length= 48
   |  geotag_ver= 2
   |  geotag_pad= 0
   |  geotag_len= 48
   |  present= GPSFlags+Latitude+Longitude+Altitude+Altitude_g+GPSTime+FractionalTime
+eph+epv+ept
   |  GPSFlags= Manual Input
   |  Latitude= 19.1234567
   |  Longitude= -155.7654321
   |  Altitude= 200.1230
   |  Altitude_g= 2.1000
   |  GPSTime= Tue, 02 Nov 2010 17:58:39 UTC (1288720719)
   |  FractionalTime= 0.100000000
   |  eph= 27.000000
   |  epv= 71.300000
   |  ept= 0.000005000
###[ 802.11 ]###
     subtype= beacon
     type= Management
     proto= 0
     FCfield=
     ID= 0
     addr1= ff:ff:ff:ff:ff:ff
     addr2= 00:01:02:03:04:05
     addr3= 00:01:02:09:01:03
     SC= 0
     addr4= None
###[ 802.11 Beacon ]###
        timestamp= 0
        beacon_interval= 100
        cap=
###[ 802.11 Information Element ]###
           ID= SSID
           len= 10
           info= 'inline-3.2'
```

Scapy decoding of GPS-tag in test packet



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	3com_03:04:05	Broadcast	802.11	108	Beacon

```
▷ Frame 1: 108 bytes on wire (864 bits), 108 bytes captured (864 bits)
▽ PPI version 0, 60 bytes
    Version: 0
  ▷ Flags: 0x00
    Header length: 60
    DLT: 105
  ▽ GPS: Lat:19.123457  Lon:-155.765432  Alt:200.123000  Alt_g:2.100000
      Header revision: 2
      Header pad: 0
      Header length: 48
    ▷ Present: 0x000003ff
    ▷ GPSFlags: 0x00000080
      Latitude: 19.1234567
      Longitude: -155.7654321
      Altitude: 200.123
      Altitude_gnd: 2.1
      GPSTimestamp: Nov  2, 2010 17:58:39.100000000 UTC
      Horizontal Error (m): 27
      Vertical Error (m): 71.3
      Time Error (s): 5e-06
▷ IEEE 802.11 Beacon frame, Flags: ........
▷ IEEE 802.11 wireless LAN management frame
```

Wireshark decoding of GPS-TAG in test packet

4. VECTOR-TAG

The VECTOR-TAG allows applications to encode vectors in 3-dimensional space. Vectors can provide offsets and rotations. This allows them to precisely describe an arbitrary point and orientation. The combination of position and orientation is called a Reference Frame. Details on the coordinate systems used to describe these reference frames are presented in section 8.

Bits that influence the mathematical interpretation of VECTOR-TAGS are stored in the VectorFlags bitmask. Bits that characterize what a VECTOR-TAG is describing are stored in the VectorCharacteristics bitmask. These bits describe what the vector represents: direction of travel, orientation of antenna, and so on. While they may influence how an application interprets a particular VECTOR tag, they do not influence the mathematical orientation.

While the VECTOR-TAG can store a rich amount of rotational information, rotation around the Z-axis will probably be the most useful to applications; when stored in absolute values this corresponds to degrees rotated around true (not magnetic) North.

Bit (LSB=0)	**Length**	**Name**	**Encoding** (unit, format, coordinate-sys)
0	4	VectorFlags	32-bit bitmask, details below
1	4	VectorCharacteristics	32-bit bitmask, details below
2	4	Pitch (Rot-X)	degrees, fixed3_6, `E/N/U`, `R/F/U`
3	4	Roll (Rot-Y)	degrees, fixed3_6, `E/N/U`, `R/F/U`
4	4	Heading (Rot-Z)	degrees, fixed3_6, `E/N/U`, `R/F/U`
5	4	Off-X: (East/Right)	meters, fixed6_4, `E/N/U`, `R/F/U`
6	4	Off-Y: (North/Forward)	meters, fixed6_4, `E/N/U`, `R/F/U`
7	4	Off-Z: (up)	meters, fixed6_4, `E/N/U`, `R/F/U`
8-15	0	Reserved	
16	4	Err-Rot	degrees, fixed3_6
17	4	Err-Off	meters, fixed6_4
18-27	0	Reserved	
28	32	DescriptionString	32 byte NUL padded ASCII description string.
29	4	AppId	See section 2.2.3.1
30	60	AppData	See section 2.2.3.2
31	0	Indicates extended bitmap.	

As currently defined a valid VECTOR-TAG shall not be any larger than **144** bytes (the size of a `base_geotag_header` plus every currently defined field), or less than 8 bytes (the size of a `base_geotag_header`). Any GEOLOCATION-TAG that does not include at least a `base_geotag_header` is invalid.

4.1. VectorFlags bitmask

The VectorFlags bitmask defines how to interpret the values in a vector tag. There are currently only 3 bits defined. Bit 0 is used to mark a vector as the Forward frame of reference. This allows other vectors further down the packet to provide offsets and rotations relative to this vector. This bit is designated as `DEFINES_FORWARD_FRAME OF_REFERENCE`.

Bits 1 and 2 are combined into a subfield known as RelativeTo. The combination of these two bits is used to indicate which Reference Frame this Vector is RelativeTo.

Bit	Name	Description
0	DefinesForward	Indicates that the Frame of Reference defined by this VECTOR-TAG is the Forward Frame of Reference. The Forward Frame of Reference is the basis of the relative coordinate system described in Section 8.2, and more information on the Forward Frame of Reference is provided in Section 8.3.2.
1,2	RelativeTo	00: All rotations and offsets are relative to the **Forward** frame of reference. (0x00) 01: All rotations and offsets are relative to the **Earth** frame of reference. (0x01) 10: All rotations and offsets are relative to the **Current** Frame of reference. (0x02) 11: Reserved See Section 8 for details on the definition of each reference frame, as well as the ENU and RFU coordinate systems utilized in each.
3-31	Reserved	

4.2. *VectorCharacteristics bitmask*

The VectorCharacteristics field is used to help applications describe what the encoded VECTOR is supposed to indicate. The bits here are purely descriptive; they do not change how any VECTOR-TAG should be handled mathematically.

Bit	Name	Description
0	ANTENNA	This vector indicates the direction of an antenna
1	DIRECTION_OF_TRAVEL	This vector indicates the direction of travel of a vehicle
2	FRONT_OF_VEHICLE	Indicates the direction of the front of a vehicle. May differ from actual direction of travel in some cases (boat drifting, car in reverse, etc)
3	ANGLE_OF_ARRIVAL	Vector indicates the angle-of-arrival of a packet. This vector may be synthesized from many different data points at the application level. It is **not** equivalent to antenna direction.
4	TRANSMITTER_POSITION	This is combined with a vector offset to allow applications to explicitly encode where they think a transmitter is located. See 10.10 for example.
5-7	Reserved	
8	GPS_DERIVED	Vector was derived from GPS hardware in some manner (most likely, heading is based off of deltas in position)
9	INS_DERIVED	Vector was derived from INS system
10	COMPASS_DERIVED	Vector was derived from digital compass
11	ACCELEROMTER_DERIVED	Vector was derived from accelerometer(s)
12	HUMAN_DERIVED	Vector was derived from manual user input
13-31	Reserved	

Note that these values are not mutually exclusive. For example, an antenna pointed out the windshield of a car could set bits 0, 1, and 2 (Provided the car was not in reverse, which would invalidate the DIRECTION_OF_TRAVEL bit). A VECTOR-TAG could also contain information derived from both a GPS and digital compass, for example.

4.3. *Vector operations*

All vector operations take place on a specific reference frame. The reference frame that the offsets and rotations are applied to is defined by the value in the RelativeTo subfield of VectorFlags. There are three possible options:

4.3.1. RelativeTo: Earth

Rotations RelativeTo Earth can be though of as absolute rotations. Since these rotations are on the East/North/Up coordinate system, they provide an anchor for all relative rotations to build off. Applications that encounter a vector that is

RelativeTo:Earth will not need information contained in any other vector to convert it to world coordinates. (A 45.0 degree Rot-Z rotation that is RelativeTo Earth corresponds to North-East).

4.3.2. RelativeTo: CurrentFrame

Some applications may find it useful to chain offsets or rotations together. In this case they can use RelativeTo:CurrentFrame. As we will see below, CurrentFrame is a frame of reference that represents the most recently processed Vector. This means that Vector tags that set RelativeTo:CurrentFrame will be applied to the most recent previously processed Vector tag.

4.3.3. RelativeTo: ForwardFrame

Finally, the PPI-GEOLOCATION specification allows applications to indicate that a given vector defines the *Forward* Frame of reference. This is useful in advanced applications, which need to encode a system of rotations relative to some common reference frame other than Earth.

For example, a system consisting of two antennas (one out the passenger and driver side door) could conveniently be encoded by saying they are 90 and 270 degrees rotated relative to the forward frame of reference; In this case the application would need to provide a vector indicating the front of the vehicle is Forward. (This exact scenario is covered in detail in 10.4).

4.4. Creating Reference Frames.

While the previous section illustrates how a vector indicates what reference frame it should be applied to, Vectors are also used to create Reference Frames. The precise rules for setting the values in a given reference frame are specified in the rules that define the PPI-GEOLOCATION state machine (section 9). The following summary provides a quick understanding of how the key Reference Frames are initialized and updated.

4.4.1. Earth Frame of reference

The Earth Frame of Reference is centered at a provided Latitude, Longitude, Altitude, and matches standard map coordinates, where the X, Y, and Z axes align with East, North, and Up, respectively. This reference frame is described in detail in 8.1. Rotations and offsets that are RelativeTo: Earth take place on the East/North/Up coordinate system.

4.4.2. Current Frame of reference

The current frame of reference is updated upon successful processing of any Vector tag. Therefore the current frame of reference contains the most recently processed Vector tag.

4.4.3. Forward Frame of reference

The Forward frame of reference is updated upon successful processing of any Vector tag, which has set the **`DEFINES_FORWARD`** (`BIT 0`) in VectorFlags. The ability for an application to define its own ReferenceFrame allows advanced applications to store data in whatever local coordinate system is convenient.

4.4.4. Reference Frame summary

Vectors that are RelativeTo the EarthFrame of reference utilize the East/North/Up (E/N/U) coordinate system described in 8.1. An application that reads rotational information in the E/N/U format does not need information about any other frame of reference (such as the heading a car was moving at) to determine the orientation of a vector.

Vectors that are RelativeTo the CurrentFrame of reference, or RelativeTo the Forward frame of reference utilize the Right/Forward/Up (R/F/U) coordinate system described in 8.2.

All operations (either offsets or rotations) take place on the reference frame indicated by the RelativeTo subfield of VectorFlags.

4.5. Rotations.

VECTOR-TAGS may contain zero or more rotations which take place on the ReferenceFrame indicated by the RelativeTo: subfield of VectorFlags. The appropriate rotation matrices are provided in 8.6.

4.6. Offsets

VECTOR-TAGS may contain zero or more offsets from which take place on the ReferenceFrame indicated by the RelativeTo: subfield of VectorFlags.

4.7. Description

All currently defined GEOLOCATION-TAGS include a 32-byte, NULL padded, ASCII description string. Examples for vectors include "VecforAnt1", "Constant offset for ANT1" or "DOT provided by GPS", and so on.

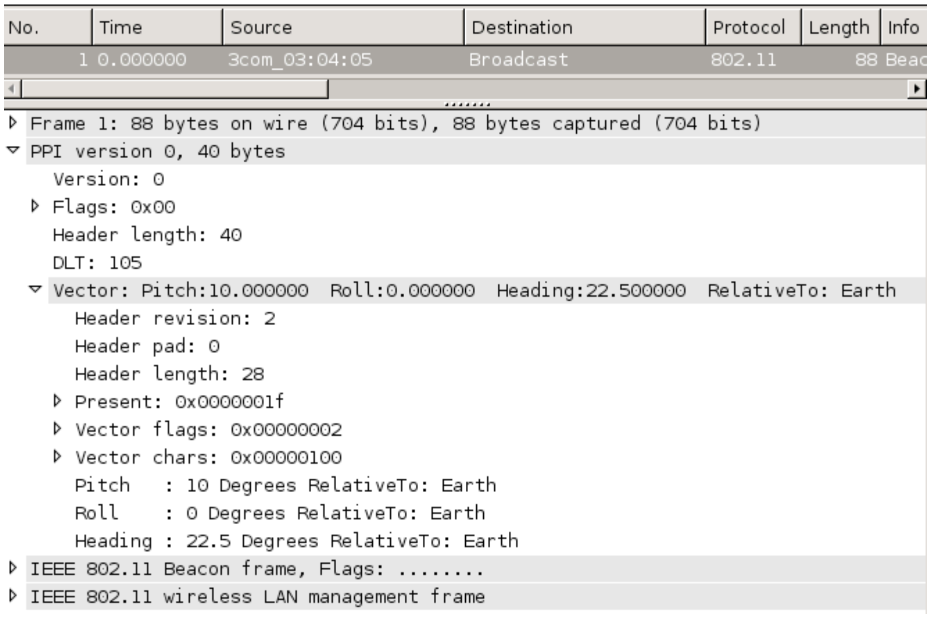
4.8. ApplicationData

All currently defined GEOLOCATION-TAGS include a 60-byte field for arbitrary application level data. See section 2.2.3 for details.

4.9. VECTOR-TAG example

A concrete example of the VECTOR-TAG encoding scheme is shown below.

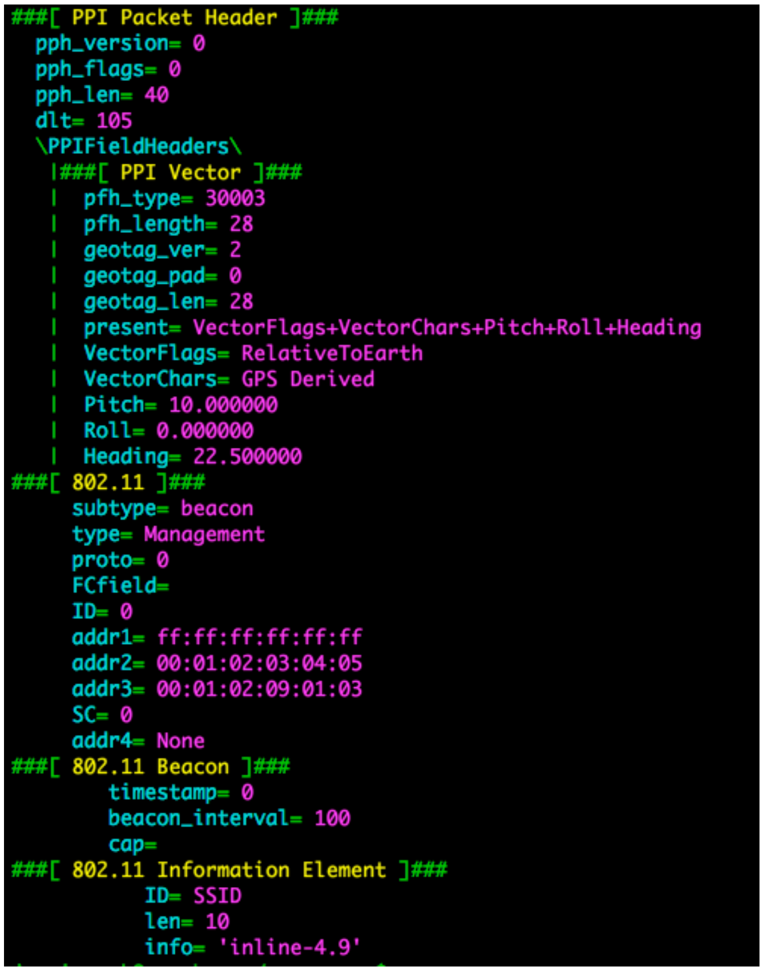
VECTOR-TAG example		
Field Name	**Native Value**	**Encoded** (little endian)
VectorFlags	DEFINES_FORWARD:0 RelativeTo:**Earth**	0x0200000
VectorChars	GPS_DERIVED	0x00010000
Rot-X (pitch)	10.0 (Slightly upward)	0x80969800
Rot-Y (roll)	0.0	0x00000000
Rot-Z (heading)	22.5 (NNE)	0xa0525701



```
No.      Time        Source           Destination      Protocol  Length  Info
       1 0.000000    3com_03:04:05    Broadcast        802.11         88 Beac

▷ Frame 1: 88 bytes on wire (704 bits), 88 bytes captured (704 bits)
▽ PPI version 0, 40 bytes
    Version: 0
  ▷ Flags: 0x00
    Header length: 40
    DLT: 105
  ▽ Vector: Pitch:10.000000  Roll:0.000000  Heading:22.500000  RelativeTo: Earth
      Header revision: 2
      Header pad: 0
      Header length: 28
    ▷ Present: 0x0000001f
    ▷ Vector flags: 0x00000002
    ▷ Vector chars: 0x00000100
      Pitch   : 10 Degrees RelativeTo: Earth
      Roll    : 0 Degrees RelativeTo: Earth
      Heading : 22.5 Degrees RelativeTo: Earth
▷ IEEE 802.11 Beacon frame, Flags: ........
▷ IEEE 802.11 wireless LAN management frame
```

Wireshark decoding of VECTOR-TAG in test packet



```
###[ PPI Packet Header ]###
  pph_version= 0
  pph_flags= 0
  pph_len= 40
  dlt= 105
  \PPIFieldHeaders\
   |###[ PPI Vector ]###
   |  pfh_type= 30003
   |  pfh_length= 28
   |  geotag_ver= 2
   |  geotag_pad= 0
   |  geotag_len= 28
   |  present= VectorFlags+VectorChars+Pitch+Roll+Heading
   |  VectorFlags= RelativeToEarth
   |  VectorChars= GPS Derived
   |  Pitch= 10.000000
   |  Roll= 0.000000
   |  Heading= 22.500000
###[ 802.11 ]###
     subtype= beacon
     type= Management
     proto= 0
     FCfield=
     ID= 0
     addr1= ff:ff:ff:ff:ff:ff
     addr2= 00:01:02:03:04:05
     addr3= 00:01:02:09:01:03
     SC= 0
     addr4= None
###[ 802.11 Beacon ]###
        timestamp= 0
        beacon_interval= 100
        cap=
###[ 802.11 Information Element ]###
           ID= SSID
           len= 10
           info= 'inline-4.9'
```

Scapy decoding of VECTOR-TAG in test packet

5. SENSOR-TAG

The Sensor-TAG allows applications to encode commonly available sensor information such as velocity and acceleration in a consistent manner. All Sensor tags can express values in three-dimensions, which is stored with the appropriate frame of reference. Sensor tags can also encode total values as well as error levels.

Sensor types that are inherently dimensionless (Temperature, etc) should be stored in the Val_T field.

Bit (LSB=0)	**Length**	**Name**	**Encoding** (unit, format)
0	2	SensorType	16-bit identifier (see table below)
1	1 (**SIGNED**)	ScaleFactor	Signed byte, -128 to +127 range. All values multiplied by 10^scale_factor, 10^{-128} to 10^{127}
2	4	Val_X	Fixed 6_4, X-Axis component
3	4	Val_Y	Fixed 6_4, Y-Axis component
4	4	Val_Z	Fixed 6_4, Z-Axis component
5	4	Val_T	Fixed 6_4, Total magnitude.
6	4	Val_E	Fixed 6_4, Error term
7-27	0	Reserved	
28	32	DescriptionString	32 byte NUL padded ASCII description string.
29	4	AppId	See section 2.2.3.1
30	60	AppData	See section 2.2.3.2
31	0	Indicates extended bitmap.	

5.1. *Sensor tag types*

A list of discrete values that can be used for SensorType, and the associated units for each val_X/Y/Z/T field is provided below

Value	**Name**	**Units**
0	Reserved	Reserved
1	Velocity	Meters/sec
2	Acceleration	Meters/sec/sec
3	Jerk	Meters/sec/sec/sec
4-99	Reserved	
100	Rotation	Degress/sec
101	Magnetic	Tesla
102-999	Reserved	
1000	Temperature	Celsius
1001	Barometer	Pascal
1002	Humidity	Percent
1003-1999	Reserved	
2000	CLOCK (**TDOA**)	Seconds (Offset from GPSTime)
2001	Phase	Degrees (0-360)
2001+	Reserved	

As currently defined a valid SENSOR-TAG shall not be any larger than **127** bytes (the size of a `base_geotag_header` plus every currently defined field), or less than 8 bytes (the size of a `base_geotag_header`). Any GEOLOCATION-TAG that does not include at least a `base_geotag_header` is invalid.

5.2. *Sensor tag notes*

All applications which utilize the TDOA_CLOCK or Phase sensor **must** include an AppId. A more detailed explanation of these sensors is given in 10.8 and 10.9.

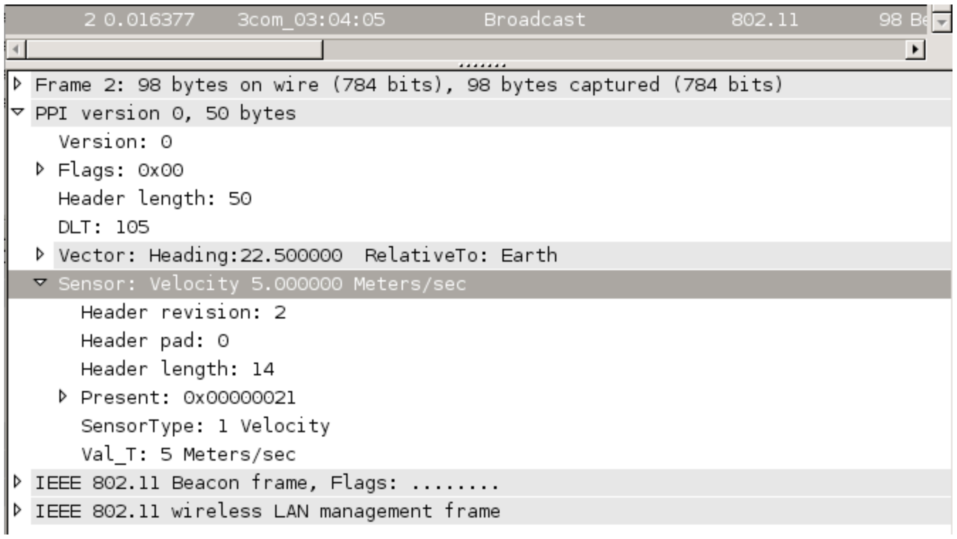
5.3. *Sensor tag example (total velocity)*

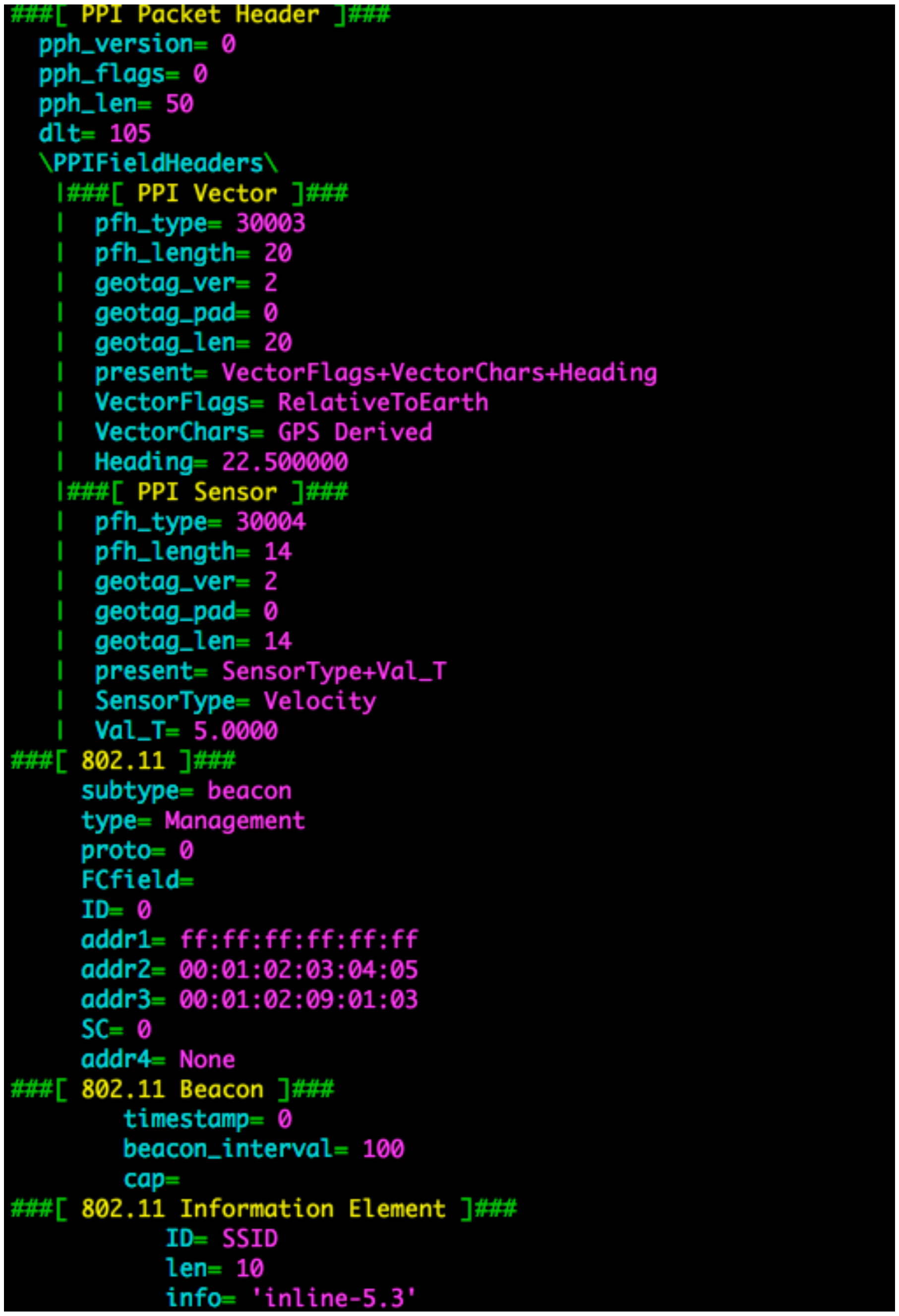
The following example illustrates how to encode a total velocity of North by North East at 5m/s. This would reflect real-world usage of a speed provided from a GPS.

VECTOR-TAG example 5.3		
Field Name	**Native Value**	**Encoded** (little endian)
VectorFlags	DEFINES_FORWARD:0 RelativeTo:**Earth**	0x0200000
VectorChars	GPS_DERIVED	0x00010000
Rot-Z (heading)	22.5 (NNE)	0xa0525701

SENSOR-TAG Example 5.3		
Field Name	**Native Value**	**Encoded** (little endian)
SensorType	Velocity (0x0001)	0x0100
CcVal-T	5.0	0x50954a6b

Detailed examples illustrating velocity and acceleration are provided in 10.3. An example illustrating the use of the TDOA-CLOCK field is given in 10.8.


```
2 0.016377    3com_03:04:05        Broadcast        802.11     98 Be
Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
PPI version 0, 50 bytes
  Version: 0
  Flags: 0x00
  Header length: 50
  DLT: 105
  Vector: Heading:22.500000  RelativeTo: Earth
  Sensor: Velocity 5.000000 Meters/sec
    Header revision: 2
    Header pad: 0
    Header length: 14
    Present: 0x00000021
    SensorType: 1 Velocity
    Val_T: 5 Meters/sec
IEEE 802.11 Beacon frame, Flags: ........
IEEE 802.11 wireless LAN management frame
```
Wireshark decoding of SENSOR-TAG in test packet



```
###[ PPI Packet Header ]###
  pph_version= 0
  pph_flags= 0
  pph_len= 50
  dlt= 105
  \PPIFieldHeaders\
   |###[ PPI Vector ]###
   |  pfh_type= 30003
   |  pfh_length= 20
   |  geotag_ver= 2
   |  geotag_pad= 0
   |  geotag_len= 20
   |  present= VectorFlags+VectorChars+Heading
   |  VectorFlags= RelativeToEarth
   |  VectorChars= GPS Derived
   |  Heading= 22.500000
   |###[ PPI Sensor ]###
   |  pfh_type= 30004
   |  pfh_length= 14
   |  geotag_ver= 2
   |  geotag_pad= 0
   |  geotag_len= 14
   |  present= SensorType+Val_T
   |  SensorType= Velocity
   |  Val_T= 5.0000
###[ 802.11 ]###
     subtype= beacon
     type= Management
     proto= 0
     FCfield=
     ID= 0
     addr1= ff:ff:ff:ff:ff:ff
     addr2= 00:01:02:03:04:05
     addr3= 00:01:02:09:01:03
     SC= 0
     addr4= None
###[ 802.11 Beacon ]###
        timestamp= 0
        beacon_interval= 100
        cap=
###[ 802.11 Information Element ]###
           ID= SSID
           len= 10
           info= 'inline-5.3'
```

Scapy decoding of SENSOR-TAG in test packet

6. ANTENNA-TAG

The ANTENNA-TAG is designed to capture as much fundamental knowledge about an antenna as possible, while allowing applications the flexibility they need to expand upon it in the future. While only a handful of fields are supported (gain, beamwdith, modelname, etc), by unambiguously defining characteristics present in every antenna, a minimum level of interoperability can be achieved. The ANTENNA-TAG also provides applications the ability to encode application-specific antenna information as well.

Like all GEOLOCATION-TAGS, the ANTENNA-TAG begins with a `geotag_base_ header`. The following table describes the bitmask to be used with this tag.

Bit (LSB=0)	**Length**	**Name**	**Encoding** (unit, format)
0	4	AntennaFlags	Bitmask, 32-bits
1	1	Gain	dBi, unsigned 8-bit
2	4	Horizontal beamwidth	degrees, `fixed3_6`
3	4	Vertical beamwidth	degrees, `fixed3_6`
4	4	PrecisionGain	dBi, fixed3_6
5	2	BeamID	Identifies beam pattern (electronically steerable antennas)
6-25	0	Reserved	
26	32	SerialNumber	ASCII only, null padded
27	32	Modelname	ASCII only, null padded
28	32	DescriptionString	ASCII only, null padded
29	4	AppId	See section 2.2.3.1
30	60	AppData	See section 2.2.3.2
31	0	Indicate extended bitmap.	

As currently defined a valid ANTENNA-TAG shall not be any larger than **187** bytes (the size of a `base_geotag_header` plus every currently defined field)**,** or less than 8 bytes (the size of a `base_geotag_header).` Any GEOLOCATION-TAG that does not include at least a `base_geotag_header` is invalid.

6.1. AntennaFlags Bitmask

The AntennaFlags bitmask is a 32-bit bitmask indicating various attributes about the current antenna and its configuration. There are currently only seven bits defined.

Bit	Description
0	Antenna part of MIMO system
1	Horizontally polarized
2	Vertically polarized
3	Circularly polarized (left handed)
4	Circularly polarized (right handed)
5-15	Reserved
16	Electronically steerable antenna
17	Mechanically steerable antenna
18-31	Reserved

6.2. ANTENNA-TAG fields

Name	Description
AntennaFlags	A 32-bit bitmask indicating various attributes about the current antenna and its configuration. Most useful is polarization and if the current ANTENNA is part of a MIMO system.
Gain	Unsigned 8-bit value, dBi. **Not** encoded in any fixed point format. (Ranges from 0-255)
HorizBW	Horizontal beamwidth of antenna expressed in degrees. `Fixed3_6`
VertBW	Vertical beamwidth of antenna expressed in degrees. `Fixed3_6`
PrecisionGain	Gain in dBi, expressed as a fixed3_6 (allows expression of partial dB). Takes priority over Gain field when both are present.
BeamId	Electronically steerable antennas have a finite set of unique beam patterns. This field encodes exactly which pattern was being used. 16-bit, little endian. Antenna-specific.
SerialNumber	32 bytes, fixed length, null padded ASCII string. Serial Number of the Antenna. Examples include 00000001.
Modelname	32 bytes, fixed length, null padded ASCII string. Applications should encourage the encoding of a model name unique to a manufacturer. Good examples include: OD9-8, PA24-13, DC24HDPF1PF-EZ, SA24-120-16-WB
DescriptionString	32-byte NULL padded ASCII description of what this ANTENNA-TAG is doing. Examples include "Out passenger side door" or "Stationary-antenna-1"
AppId	See section 2.2.3.1
AppData	See section 2.2.3.2

As currently defined a valid ANTENNA-TAG shall not be any larger than **187** bytes (the size of a `base_geotag_header` plus every currently defined field)**,** or less than 8 bytes (the size of a `base_geotag_header).` Any GEOLOCATION-TAG that does not include at least a `base_geotag_header` is invalid

6.3. ANTENNA-TAG usage rules.

Although ANTENNA-TAGs only contain a handful of fields, by following the following rules a significant amount of information about the connected antenna can be discerned.

6.3.1. Omni-directional antennas:

ANTENNA-TAG producers should set the HorizontalBeamwidth to 360.0 (360000000, or 0x002a7515 (little endian)) for any omni-directional antenna. Similarly, consumers interested in discerning between Omni-directional and directional antennas should test the HorizontalBeamwidth field for equality with 360.0.

If ANTENNA-TAG producers are observed to be not honoring the 360.0 = Omni-directional convention, it is permissible for consumers to characterize any antennas with HorizBeamwidth >= 270.0 degrees as an Omni.

6.3.2. Modelname:

Applications producing model names should try to encode a uniquely identifying model name into the Modelname. If an application is unable to provide a unique Modelname, it may generate a generic Modelname using the following scheme.

Generic modelnames shall be of the form:

[*GainInDBI*dBi-][Internal|Omni|MagMountOmni|Panel|Yagi|Sector|Grid]
where *GainInDBI* is apparent. **Consumers of Modelnames shall NOT parse them for gain values.** Producers of Modelnames will NOT include the gain portion in the Modelname unless the actual gain field is also present.

Example generic Modelnames include (but are not limited to):
5dBi-Omni, 8.5dBi-MagMountOmni, 27dBi-Yagi, 15dBi-Panel, MagMountOmni, Yagi, Grid.

6.3.3. Fractional gain:

Many antennas have fractional amount of gain (e.g. 8.5dBi). Applications that need to encode gain with such precision are encouraged to use both the gain field, as well as the PrecisionGain field (which takes precedence). Applications should round 0.5 **up** when filling in the gain field.

6.3.4. BeamID field:

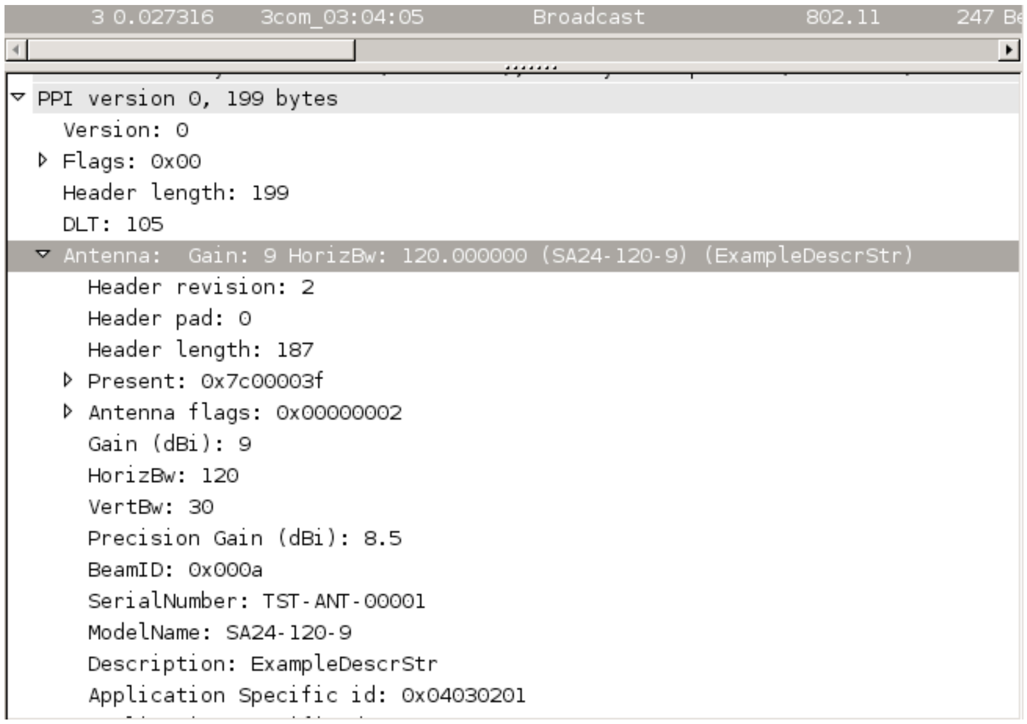
The BeamID contains vendor-specific values, placed into a standardized field. Manufacturers of electronically steerable antennas should encode the relevant orientation, beamwidth, gain, etc for a given BeamID in the appropriate fields. Vendor aware applications may be able to interpret (BeamID, ModelStr) or (BeamId, AppId) data with more fidelity than offered by standardized GEOLOCATION-TAG data.

Applications are discouraged from simply producing ANTENNA-TAGs with only BeamIDs (essentially utilizing BeamID as a magic number to identify vendor-specific antenna characteristics). Applications making use of BeamID should fill out a VECTOR tag with a relative heading and the gain/horizbw field of the associated ANTENNA tag.

6.4. ANTENNA-TAG example

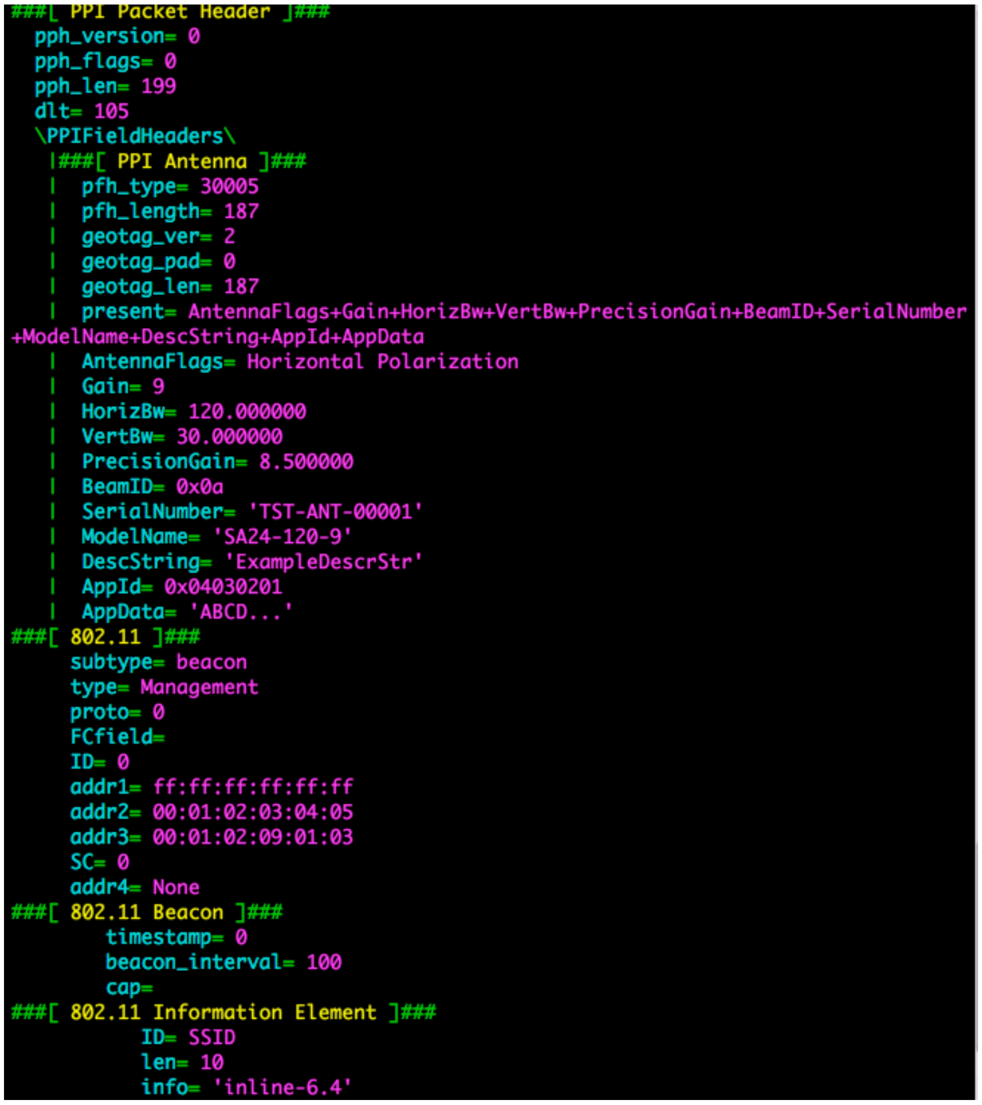
The following table provides a concrete example of ANTENNA-TAG encoding. Note that this example includes every field for the sake of completeness. The majority of applications would include only the gain and horizontal beamwidth fields.

ANTENNA-TAG Example		
Field Name	**Native Value**	**Encoded** (little endian)
AntennaFlags	Horizontal Polarity, Electronically Steerable	0x02000000
Gain	9 dBi	0x09
HorizBW	120.0	0x00E22707
VertBW	30.0	0x80C3C901
PrecisionGain	8.5dBi	0x20b38100
BeamID	10	0x0A00
SerialNumber	TST-ANT-00001	"TST-ANT-00001"
Modelname	SA24-120-9	"SA24-120-9_E"
DescriptionString	ExampleDescrStr	"ExampleDescrStr"
ApppId	0x04030201	0x01020304
AppData		"0x41424344.."



```
    3 0.027316    3com_03:04:05        Broadcast          802.11      247 B
▽ PPI version 0, 199 bytes
    Version: 0
  ▷ Flags: 0x00
    Header length: 199
    DLT: 105
  ▽ Antenna:  Gain: 9 HorizBw: 120.000000 (SA24-120-9) (ExampleDescrStr)
      Header revision: 2
      Header pad: 0
      Header length: 187
    ▷ Present: 0x7c00003f
    ▷ Antenna flags: 0x00000002
      Gain (dBi): 9
      HorizBw: 120
      VertBw: 30
      Precision Gain (dBi): 8.5
      BeamID: 0x000a
      SerialNumber: TST-ANT-00001
      ModelName: SA24-120-9
      Description: ExampleDescrStr
      Application Specific id: 0x04030201
```

Wireshark decoding of ANTENNA-TAG in packet



```
###[ PPI Packet Header ]###
  pph_version= 0
  pph_flags= 0
  pph_len= 199
  dlt= 105
  \PPIFieldHeaders\
   |###[ PPI Antenna ]###
   |  pfh_type= 30005
   |  pfh_length= 187
   |  geotag_ver= 2
   |  geotag_pad= 0
   |  geotag_len= 187
   |  present= AntennaFlags+Gain+HorizBw+VertBw+PrecisionGain+BeamID+SerialNumber
+ModelName+DescString+AppId+AppData
   |  AntennaFlags= Horizontal Polarization
   |  Gain= 9
   |  HorizBw= 120.000000
   |  VertBw= 30.000000
   |  PrecisionGain= 8.500000
   |  BeamID= 0x0a
   |  SerialNumber= 'TST-ANT-00001'
   |  ModelName= 'SA24-120-9'
   |  DescString= 'ExampleDescrStr'
   |  AppId= 0x04030201
   |  AppData= 'ABCD...'
###[ 802.11 ]###
     subtype= beacon
     type= Management
     proto= 0
     FCfield=
     ID= 0
     addr1= ff:ff:ff:ff:ff:ff
     addr2= 00:01:02:03:04:05
     addr3= 00:01:02:09:01:03
     SC= 0
     addr4= None
###[ 802.11 Beacon ]###
        timestamp= 0
        beacon_interval= 100
        cap=
###[ 802.11 Information Element ]###
           ID= SSID
           len= 10
           info= 'inline-6.4'
```

scapy decoding of ANTENNA-TAG in test packet

6.5. *ANTENNA-TAG caveats.*

While the ANTENA-TAG can convey basic properties about antennas, accurately describing the characteristics of a specific antenna requires knowing its radiation pattern. In theory applications could store their own models of antennas and utilize the Modelname to identify particular antennas in their internal database.

This would work well within a single enterprise, but without a centralized Modelname authority collisions/disparities on data captured in the wild will quickly mount. Organizations are free to try and use the application specific data field to try and minimize this problem.

The AntennaFlags bitmask has quite a bit of room to grow. If readers have plausible uses for the other bits the author would like to know.

7. RADIOTAP/802.11COMMON TAGS

Radiotap headers are currently the most widely supported format for storing signal strength, channel information, and other 802.11 specific characteristics in a pcap file. These headers precede the actual 802.11 header, and are specified in a pcap file by setting the DLT to DLT_IEEE802_11_RADIO (127). Radiotap headers are structurally very similar to GEOLOCATION-TAGs in that they both make use of a bitmask to indicate which fields are present. Radiotap headers are defined at www.radiotap.org [1].

802.11COMMON tags are designed to encapsulate most of the information present in a radiotap header, inside of an easier to parse PPI-TAG. This tag is defined by CACE in [2], and has been assigned a pfh_type value of 2. 802.11COMMON tags differ from GEOLOCATION-TAGS and Radiotap headers in that they are fixed length and utilize predetermined values to indicate that fields are not present (in lieu of the present bitmask used by radiotap and GEOLOCATION-TAGs).

While the PPI-GEOLOCATION specification recognizes that Radiotap headers are prevalent in the wild, formalizing their interaction with the PPI-GEOLOCATION state machine is difficult. Therefore, applications implementing the PPI-GEOLOCATION processing engine **may** decide to treat Radiotap headers as logically equivalent to a Dot11Common tag.

Non-legacy applications are heavily encouraged to utilize Dot11Common tags to store signal strength, channel information, etc. This will minimize the amount of ambiguity when the tagged data is run through different engines.

Applications that produce PPI-GEOLOCATION tagged data that do decide to utilize Radiotap to encode signal/channel information **must not** also include a Dot11Common tag in the packet. Including both creates an inherently ambiguous case in the processing engine.

The following table describes a useful subset of fields present in both 802.11COMMON tags and Radiotap headers; it is included for convenience only. The definition of both 802.11COMMON tags and Radiotap headers are outside the scope of this specification.

Name	**Description**	**Type**	**Size** (bytes)
Rate	Data rate in multiples of 500 Kbps Invalid value = 0x0000	Unsigned	2
Channel-Freq	Radiotap-formatted channel frequency, in MHz Invalid value = 0x0000	Unsigned	2
Channel-Flags	Radiotap-forrmatted channel flags:	Unsigned	2
dBm-Antsignal	RF signal power at antenna Invalid value = -128	Signed	1
dBm-Antnoise	RF noise at antenna Invalid value = -128	signed	1

8. Coordinate Systems

Collection systems can combine GPS and VECTOR tags, defined in sections 3 and 4, to precisely define the position and orientation of the system when the tagged packet was collected. Basic systems might use a GPS tag to provide the vehicle position, and a VECTOR tag to indicate the vehicle heading or direction an antenna was facing.

More advanced applications may include additional VECTOR fields to provide roll, pitch, and heading information. Multiple VECTOR tags can be combined to provide detailed information about the vehicle itself, the vehicle's direction of travel, and the relative position and orientation of multiple antennas located on the vehicle. Finally, applications can also include sensor tags to indicate velocity and acceleration information along any of the provided vectors.

To provide all of this functionality, two coordinate systems are defined: The East/North/Up Coordinate System and the Right/Forward/Up Coordinate System. The E/N/U system can be though of as an absolute coordinate system, and is covered in Section 8.1. It has axes aligned to East, North, and Up, matching standard map coordinates. The E/N/U system is used in any vector that is RelativeTo: Earth.

The Relative Coordinate System, covered in Section 8.2, has Right, Forward, and Up axes aligned to a specified absolute pitch, roll, and heading provided by a VECTOR tag. The Vector tags RelativeTo: field determines exactly which Vector to be used as a base for the R/F/U system.

Multiple VECTOR tags can be provided with a single packet, which can establish multiple relative coordinate systems. These multiple coordinate systems are referred to as Frames of Reference. An overview of reference frames is given in Section 8.3, and specific algorithms for establishing and updating these reference frames are provided in Section 9.

As mentioned in section 4.3, all vectors are RelativeTo one of three frames (Earth, Current, or Forward).

8.1. The Absolute East, North, Up (ENU) Coordinate System

The Absolute Coordinate System shown on the left in Figure 8.1 defines a local (X, Y, Z) = (East, North, Up) Cartesian system with axes aligned with standard map coordinates. This system is referred to as a local coordinate system, because while the directions East, North, and Up are nearly constant over a small area (small in Earth terms: hundreds of miles), they all change as the system moves around the globe. For example, the direction Up is horizontal at the equator, but vertical at the poles. The center of the coordinate system is defined at a specific latitude (φ), longitude (λ), and altitude, as shown on the right in Figure 8.1, and forms a plane tangent to the surface of the Earth.

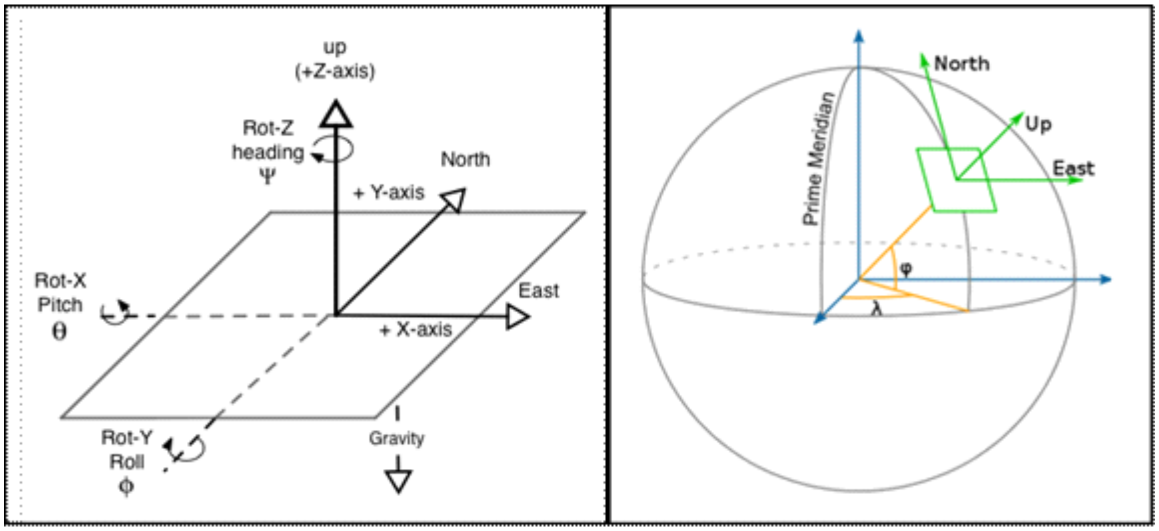


Figure 8.1: The Absolute (East, North, Up) Coordinate System.

Axis	Direction	Zero defined as:
+X	East	90 degrees clockwise from True North.
+Y	North	True North.
+Z	Up	Opposite Earth's gravitational pull.

The origin of the Absolute coordinate system is set by a GPS tag. At a minimum, this tag should provide latitude and longitude, with altitude provided or assumed to be ground level.

Once the system position is set using a GPS tag, the system orientation can be defined using a VECTOR tag. The VECTOR tag has Rot-X, Rot-Y, and Rot-Z fields, also known as Pitch, Roll, and Heading, which are shown in Figure 8.1. These absolute rotations show the system orientation as rotations about the East, North, and Up axes.

Rotation	Name	Symbol	Description
Rot-X	Pitch	θ	Counter-Clockwise rotation about the X axis. When Rot-X = 0, the front and back of vector should be parallel to the ground. When Rot-X = 90, the vector will be perpendicular to the ground, with its nose facing up.
Rot-Y	Roll	φ	Counter-Clockwise rotation about the Y axis. When Rot-Y = 0, the left and right sides of the vector should be parallel to the ground. When Rot-Y is positive, the right side of the object indicated by vector will be lower than the left side. (Right wing down convention)
Rot-Z	Heading	Ψ	Clockwise rotation about the Z axis. When Rot-Z = 0, the vector should be oriented at true (*not* magnetic) North. When Rot-Z = 90, the vector will be oriented due East. **Note**: This rotation is clockwise so the rotation angle matches a standard compass heading.

As an example, Figure 8.2 shows how these pitch, roll, and heading values describe the orientation of a vehicle, and how these values establish the Right, Forward, and Up axes of the Relative Coordinate System covered in Section 8.2.

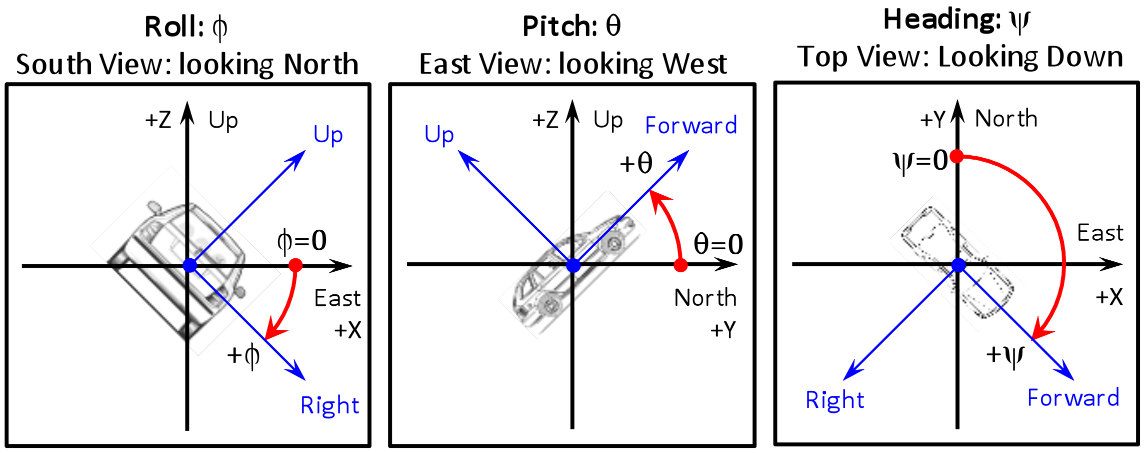


Figure 8.2: Absolute Roll, Pitch, and Heading describing Vehicle Orientation.

8.2. The Relative Right, Forward, Up (RFU) Coordinate System

The Relative Coordinate System has Right, Forward, and Up axes which track the orientation provided by a VECTOR tag. **The zeroes of the RFU system depend on what the vector has indicated it is RelativeTo:** If the Vector is RelativeTo: ForwardFrame, then the R/F/U system is aligned with the most recently processed tag which has the `FORWARD_FRAME` bit set. If the Vector is RelativeTo: CurrentFrame, then the R/F/U coordinate system is aligned with the most recently processed Vector tag. Vectors which are RelativeTo: Earth use the absolute E/N/U system.

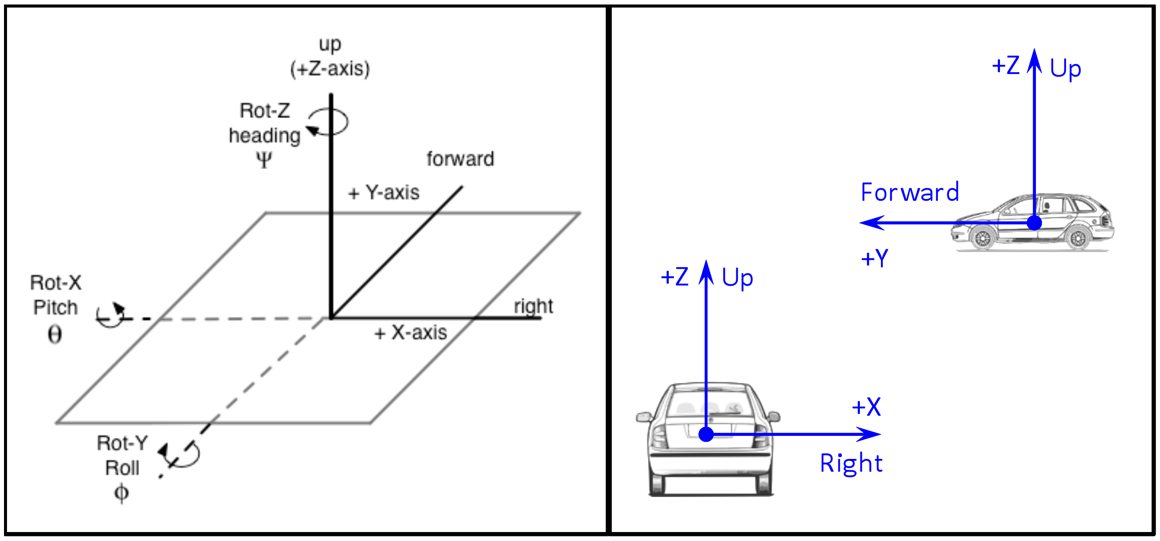


Figure 8.3: The Relative (Right, Forward, Up) Coordinate System.

For example, once a GPS tag provides the origin of the E/N/U Coordinate System described in Section 8.1, a VECTOR tag could define the orientation of a vehicle by providing Absolute Pitch, Roll, and Heading values, as shown in Figure 8.2. Once this vehicle orientation is established, setting the `FORWARD_FRAME` flag indicates that this position and orientation will be used as the origin of a Relative Coordinate system that is RelativeTo: Forward.

Figure 8.3 shows the axes of the Relative Coordinate System, as well as how the Right, Forward, and Up axes would line up on a vehicle in the example given above.

Once the relative coordinate system is established, additional VECTOR tags can provide offsets and rotations on this coordinate system. If the relative coordinate system defines the center of a vehicle, subsequent VECTOR tags may wish to define the position and orientation of one or more antennas relative to this vehicle center. Figure 8.4 shows the axes of this relative coordinate system in blue, and the start and direction of relative Pitch, Roll, and Heading rotations in red. The axes of the newly formed Frame of Reference are shown in green.

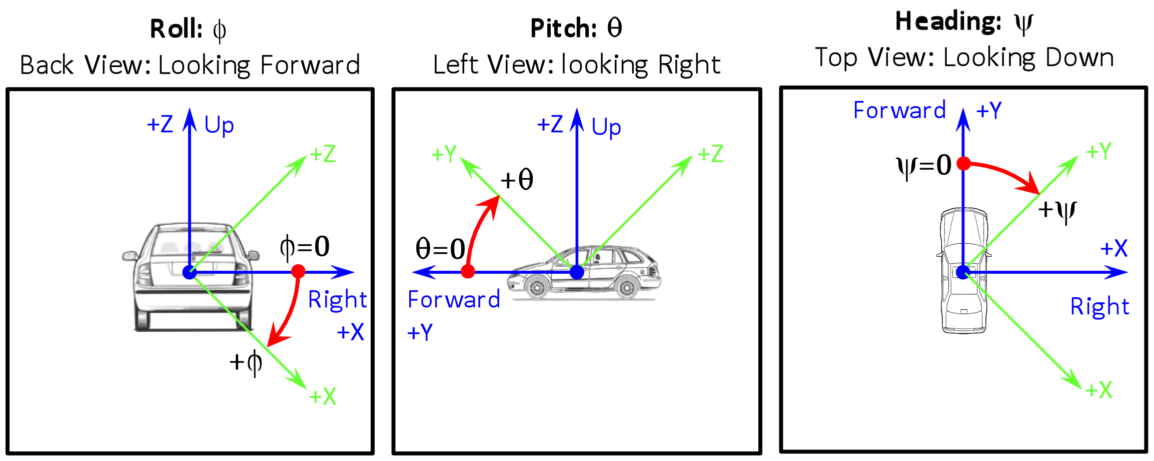


Figure 8.4: Relative Roll, Pitch, and Heading describing a new Frame of Reference.

Figure 8.2 shows how absolute rotations set the orientation of the Right, Forward, Up axes relative to the ENU coordinate system. Figure 8.4 shows how relative rotations can define a new set of axes. The current Right, Forward, Up axes are shown in blue in Figure 8.4. Relative Rotations form the new axes shown in green.

8.3. *Key Frames of reference*

A Key frame of reference is a frame of reference that Vector operations may be applied to. Currently this is limited to Earth, Current, and Forward.

8.3.1. Earth Frame of Reference

The Earth frame of reference serves as an anchor for converting relative offsets and rotations into absolute terms. The position of the Earth frame of reference is always equal to the most recently processed GPS tag. Its orientation is read-only, and uses the E/N/U coordinate system defined in 8.1. Any vector which is RelativeTo:Earth is oriented on the E/N/U coordinate system. This Reference frame was designed to be convenient for humans and digital compass's to interact with.

8.3.2. Forward Frame of Reference

The forward frame of reference is useful for applications that find it convenient to create their own local coordinate system. For example, a system of two antennas (one in the passenger and driver window), that is also equipped with a digital compass that indicates the direction of travel of the vehicle would find it convenient to encode the two antennas as 90 and 270 degrees relative to forward. In this case the application would first need to provide a vector that gives the current orientation of the vehicle. This vector would set the `DEFINES_FORWARD` (bit 0 in VectorFlags). And the following vectors indicating the orientation of the antennas would set RelativeTo: Forward. This case is covered in detail in 10.4

Different systems may choose to align forward to different system components. For example, one system may align forward to the front of the vehicle, while another may align forward to the direction of an antenna. It is important to emphasize that the `DEFINES_FORWARD` bit **is not** indicative of the front of a vehicle, or the direction of travel, or any other physical characteristic. (These physical properties are handled by the VectorCharacteristics bitmask).

The `DEFINES_FORWARD` bit is used to specify a convenient local coordinate system for an application. Applications may set this bit on multiple vectors if they desire to re-define this reference frame for any reason.

8.3.3. Current Frame of Reference

The current frame of reference is defined by the most recently processed Vector tag. It is a Key frame of reference because it allows applications to specify offsets and rotations that are relative to the most recently preceding Vector tag.

In many systems, it makes sense to define Forward as the center of the vehicle, with the X (Right) axis aligned with the right side of the vehicle, and the Y (Forward) axis aligned with the front of the vehicle. This is a good starting point to then give the relative location and orientation of antennas and other sensors. However, with electronically or mechanically steerable antennas, it is sometimes necessary to define the location of the fixed base portion of the antenna as well as the orientation of the rotating receive element.

If the base is offset as well as pitched or rolled relative to the vehicle center, it is much simpler to give that translation first, and then provide the receive element or electronic beam rotation as a second Vector relative to the base. In particular, if the antenna base contains sensors that do not rotate with the receive element, this intermediate frame needs to be defined, but marking it as Forward would make it more difficult to define the location of additional antennas or sensors. So in these cases, use a Vector RelativeTo: Forward or Earth to first define the antenna base. Then, if necessary, use Sensor tags to give any data from sensors located inside the base. Finally, define a Vector RelativeTo: CurrentFrame to give the orientation of the rotating receive element relative to the base, and set the Antenna bit in the VectorChars flag to indicate this Vector gives a new antenna location.

8.4. Non-Key Frames of Reference

Section 9 indicates that a PPI-GEOLOCATION state machine should store a Reference frame that corresponds with every VectorCharacteristic bit (Antenna, Direction of travel, Front of vehicle, etc). These Reference frames are provided to all client applications as a convenience when any state is queried. The state machine cannot be directed to perform operations on any of these frames explicitly. The state machine will only apply vector operations to Key frames of reference (Earth, Forward, Current).

8.5. Implementation of a Reference Frame

The following pseudocode demonstrates a simple implementation of a frame of reference.

```
struct GPSPos
{
        u_int32_t present;
        float lon, lat, alt
        u_int32_t GPSTime, fractionaltime;
};
struct BasicVec
{
        u_int32_t present, VectorFlags, VectorChars;
        float rotX, rotY, rotZ; //pitch, roll, heading,
                                //in absolute (E/N/U) terms
};
struct SensorData_T
{
      u_int32_t present;
      float val_X, val_Y, val_Z, val_T val_E;
      ..
struct FrameOfReference_t
{
        struct GPSPos Pos;
        struct BasicVec Vec;
        struct SensorData_T  SensorData[1024];
        //Arbitrary application specific data can
        //also   be   stored   here.   F.ex,   data   processed   from
        //ApplicationData fields, as well as vel and accel

};
FrameOfReference_t Forward_Frame;
```

A complete frame of reference would include every defined field related to position, including timestamps, error levels, rotations in 3-dimensions, and possibly sensor data such as velocity and acceleration. However parsers should accept partially defined frames of reference, producing a warning only when an operation that utilizes an undefined value is encountered. (For example, providing a vertical offset when no altitude was specified). For details on handling undefined values see Section 9.6.

8.5.1. Default Frames of Reference

It is important to note that the initial Forward and Current Frame of Reference (Defined in section 9) matches absolute ENU coordinates. In other words, the default Forward Frame and CurrentFrame is equivalent to a VECTOR tag that is RelativeTo: Earth with Rot-X, Rot-Y, Rot-Z, Offset-X, Offset-Y, and Offset-Z all set to 0. It is also important to remember that every time a GPS tag is received, all frames of reference should reset to the default state. Additional implementation details are provided in Section 9.

8.6. Converting Relative Offsets and Rotations to ENU Coordinates

Ultimately, all system information needs to be converted to ENU coordinates to allow plotting, comparison and processing of data. Section 8.6.1 describes how rotation matrices are formed from absolute and relative rotations. These rotation matrices can then be combined to translate relative rotations back to absolute terms. The resulting matrix is also used to translate relative coordinates back to ENU coordinates, as discussed in Section 8.6.2. Section 8.6.3 gives a practical application of these concepts with equations and results.

8.6.1. Computing Rotations

A relative rotation contained in a VECTOR tag can be combined with the reference frame it is RelativeTo: to give the absolute orientation of the antenna (or other object), but the absolute and relative angles usually cannot simply be added together, and the order in which the rotations are applied is critical.

If only absolute and relative heading information is provided (pitch and roll are assumed 0), then the two values can be added together directly.

This is also true for other simple cases, but in general, rotations matrices must be created representing the reference frame that the current vector is RelativeTo as well as the relative rotation.

In the following section we process a relative vector (which we assume in this example to represent an antenna) that is RelativeTo: Forward. The Forward frame of reference is used to create a rotation matrix $M_{FWD2ENU}$. The relative rotation is converted into a rotation matrix $M_{REL2FWD}$.

(If the vector had been RelativeTo:Current the math would be identical, but we would replace the values used to initialize $M_{FWD2ENU}$ with those found in CurrentFrame).

Both $M_{FWD2ENU}$ and $M_{REL2FWD}$ have the form:

$$R(\phi,\theta,\Psi)=\begin{pmatrix} \sin\phi\sin\theta\sin\Psi+\cos\phi\cos\Psi & \cos\theta\sin\Psi & -\cos\phi\sin\theta\sin\Psi+\sin\phi\cos\Psi \\ \sin\phi\sin\theta\cos\Psi-\cos\phi\sin\Psi & \cos\theta\cos\Psi & -\cos\phi\sin\theta\cos\Psi-\sin\phi\sin\Psi \\ -\sin\phi\cos\theta & \sin\theta & \cos\phi\cos\theta \end{pmatrix}$$

Where:

ϕ = Roll = Vec.RotY

θ = Pitch = Vec.RotX

Ψ = Yaw = Vec.RotZ

Therefore

$M_{FWD2ENU} = \mathrm{R}(\mathrm{Forward\_Frame.RotY}, \mathrm{Forward\_Frame.RotX}, \mathrm{Forward\_Frame.RotZ})$

$M_{REL2FWD} = \mathrm{R}(\mathrm{Rel\_Vec.RotY}, \mathrm{Rel\_vec.RotX}, \mathrm{Rel\_Vec.RotZ})$

A final rotation matrix, $M_{REL2ENU}$ is computed by multiplying $M_{REL2FWD}$ and $M_{FWD2ENU}$

$M_{REL2ENU} = M_{REL2FWD} M_{FWD2ENU}$

$M_{REL2ENU}$ transforms a point or vector from a relative frame (Rel_Vec above) into the ENU coordinate system.

The final absolute (E/N/U) rotation angles are:

$\phi_{FIN} = ROLL_{FIN} = \mathrm{atan2}(M_{REL2ENU}[2,0] M_{REL2ENU}[2,2])$

$\theta_{FIN} = PITCH_{FIN} = \mathrm{asin}(\mathrm{M}_{\mathrm{REL2ENU}}[2,1])$

$\Psi_{FIN} = HEADING_{FIN} = \mathrm{atan2}(\mathrm{M}_{\mathrm{REL2ENU}}[0,1], \mathrm{M}_{\mathrm{REL2ENU}}[1,1])$

Where $M[a,b]$ means row a, column b of M, and atan2 expects (y,x) input, not (x,y)

8.6.2. Relative Offsets

VECTOR tags may store E/N/U or R/F/U offsets to further encode the location of a particular element of the system. (E/N/U offsets are of marginal utility, but a consequence of allowing vector operations RelativeTo:Earth). R/F/U offsets can be specified from either the Current Frame or Forward Frame of reference via the RelativeTo: field. Relative offsets specified in R/F/U coordinates can be translated to ENU coordinates by storing them in matrix form and multiplying by $M_{FWD2ENU}$. An example is given in Section 8.6.3.

8.6.3. Example System with Relative Rotation and Offset Computations

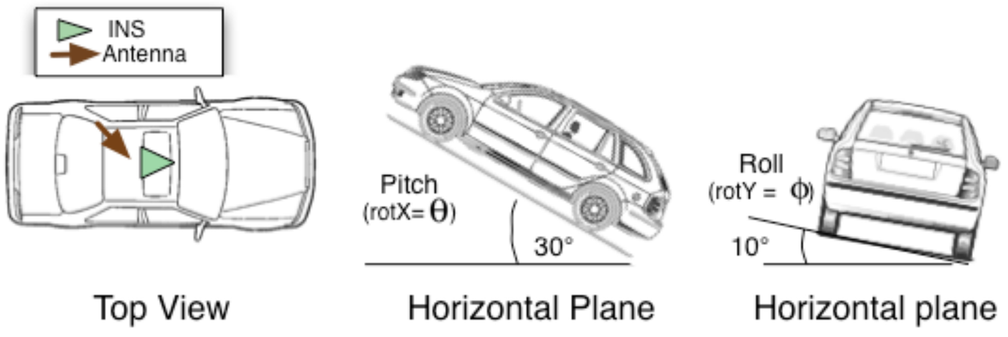
This section illustrates one typical system configuration that uses both absolute and relative rotations, as well as relative offsets to define the location of individual system elements. The locations of these elements are translated back to absolute E/N/U coordinates, using the equations provided in Section 8.6.1, and the Frame of Reference structure defined in Section 8.5.

8.6.3.1. System Definition

Please consider the following system:

- A vehicle with a 1m wide, 1.5m long roof
- An INS unit mounted at the center of the roof, providing GPS information as well as Roll, Pitch, and Heading information
- A steerable antenna mounted at the back left corner of the vehicle roof. The antenna can only rotate in the azimuth plane, and provides relative azimuth angle feedback as it rotates, and started (0 degree mark) facing the front of the vehicle.

Figure 8.5: Example System Orientation and Component Layout



8.6.3.2. System GEOLOCATION Tags

When a packet is received, the system would output one GPS and two VECTOR tags. The example below shows the system output if the vehicle was travelling East up a steep hill with the antenna pointing towards the right front corner of the vehicle roof:

GPS-TAG	
Latitiude	40.787743°
Longitude	-73.971210°
Altitude	200.123m

VECTOR-TAG	
VectorFlags	**0x03: DefinesForward + RelativeTo:Earth** `Bit 0:` Vector defines the Forward Frame of Reference `Bit1,2: RelativeTo:`**`Earth`**

VectorCharacteristics	`0x06`
	`FRONT_OF_VEHICLE`
	`DIRECTION_OF_TRAVEL` (vector indicates direction of travel)

Rot-X (Pitch)	30.0°
Rot-Y (Roll)	10.0°
Rot-Z (Heading)	90.0°
Off-R (Right)	0.0
Off-F (Forward)	0.0
Off-U (Up)	0.0

VECTOR-TAG	

VectorFlags	`0x00`
	`RelativeTo: Forward`

VectorCharacteristics	0x01 `ANTENNA` (vector represents an antenna)
Heading (RotZ)	45.0°
Off-R (Right)	-0.5 meters
Off-F (Forw)	-0.75 meters

The VectorCharacteristics flags of the first VECTOR-TAG indicate this tag provides the vehicle direction of travel as well as the orientation of the front of the vehicle. The VectorFlags indicate that the rotations provided are RelativeTo: Earth, meaning they give the orientation of the vehicle in the E/N/U coordinate system. This tag also indicates it should be used as the Forward frame of reference by setting the `DEFINES_FORWARD_FRAME_OF_REFERENCE` bit.

The VectorCharacteristics and VectorFlags flags of the second VECTOR-TAG indicate this tag gives the position and orientation of an Antenna, relative to the currently defined forward frame of reference.

8.6.3.3. Calculated Rotation Matrices

Using the formula shown in 8.6.1 we can generate the following two rotation matrices.

$$M_{FWD2ENU} = \mathrm{R}(\mathrm{Forward\_Frame.RotY}, \mathrm{Forward\_Frame.RotX}, \mathrm{Forward\_Frame.RotZ})$$

$$M_{ANT2FWD} = \mathrm{R}(\mathrm{Antenna.RotY}, \mathrm{Antenna.RotX}, \mathrm{Antenna.RotZ})$$

$$M_{FWD2ENU} = R(10^\circ, 30^\circ, 90^\circ) = \begin{pmatrix} 0.0868 & 0.8660 & -0.4924 \\ -0.9848 & 0.0000 & -0.1736 \\ -0.1504 & 0.5000 & 0.8529 \end{pmatrix}$$

$$M_{ANT2FWD} = R(0^\circ, 0^\circ, 45^\circ) = \begin{pmatrix} 0.7071 & 0.7071 & 0.0000 \\ -0.7071 & 0.7071 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 \end{pmatrix}$$

$$M_{ANT2ENU} = M_{ANT2FWD} M_{FWD2ENU}$$

$$M_{ANT2ENU} = \begin{pmatrix} 0.7071 & 0.7071 & 0.0000 \\ -0.7071 & 0.7071 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 \end{pmatrix} \begin{pmatrix} 0.0868 & 0.8660 & -0.4924 \\ -0.9848 & 0.0000 & -0.1736 \\ -0.1504 & 0.5000 & 0.8529 \end{pmatrix}$$

$$M_{ANT2ENU} = \begin{pmatrix} -0.5510 & 0.6738 & -0.4924 \\ -0.6964 & -0.6964 & -0.1736 \\ -0.4599 & 0.2472 & 0.8529 \end{pmatrix}$$

Converting to Absolute ENU Rotations

With $M_{ANT2ENU}$ computed, we can derive the ENU rotations of the antenna.

$$\phi_{FIN} = Roll_{FIN} = \operatorname{atan}2(M_{ANT2ENU}[2,0], M_{ANT2ENU}[2,2])$$
$$= \operatorname{atan}2(-0.4599, 0.8529) = 28.3^{\circ}$$

Antenna_Absolute.RotY = 28.3

$$\theta_{FIN} = Pitch_{FIN} = \operatorname{asin}(M_{ANT2ENU}[2,1])$$
$$= \operatorname{asin}(0.2472) = 14.3^{\circ}$$

Antenna_Absolute.RotX = 14.3

$$\Psi_{FIN} = Heading_{FIN} = \operatorname{atan}2(M_{ANT2ENU}[0,1], M_{ANT2ENU}[1,1])$$
$$= \operatorname{atan}2(0.6738, -0.6964) = 135.9^{\circ}$$

Antenna_Absolute.RotZ =135.9

Which indicates that the antenna was oriented with 14.3 degrees of (absolute) pitch, 28.3 degrees of (absolute) roll, and was pointed approximately south by south east.

8.6.4. Converting offsets to ENU Coordinates

In order to convert offsets into E/N/U coordinates first construct a matrix representing the offsets IN R/F/U coordinates.

$$P_{ANT\_FWD} = \begin{pmatrix} OffX \\ OffY \\ OffZ \end{pmatrix} = \begin{pmatrix} -0.50\,\text{Right} \\ -0.75\,\text{Forw} \\ 0.00\ \ \text{Up} \end{pmatrix}$$

Convert P_{ANT_FWD} to East, North, Up coordinates, by multiplying by $M_{FWD2ENU}$

$$P_{ANT\_ENU} = M_{FWD2ENU} P_{ANT\_FWD}$$

$$P_{ANT\_ENU} = \begin{pmatrix} 0.0868 & 0.8660 & -0.4924 \\ -0.9848 & 0.0000 & -0.1736 \\ -0.1504 & 0.5000 & 0.8529 \end{pmatrix} \begin{pmatrix} -0.50\,\text{m Right} \\ -0.75\,\text{m Forw} \\ 0.00\,\text{m}\ \ \text{Up} \end{pmatrix} = \begin{pmatrix} -0.69\,\text{m East} \\ 0.49\,\text{m}\ \ \text{North} \\ -0.30\,\text{m Up} \end{pmatrix}$$

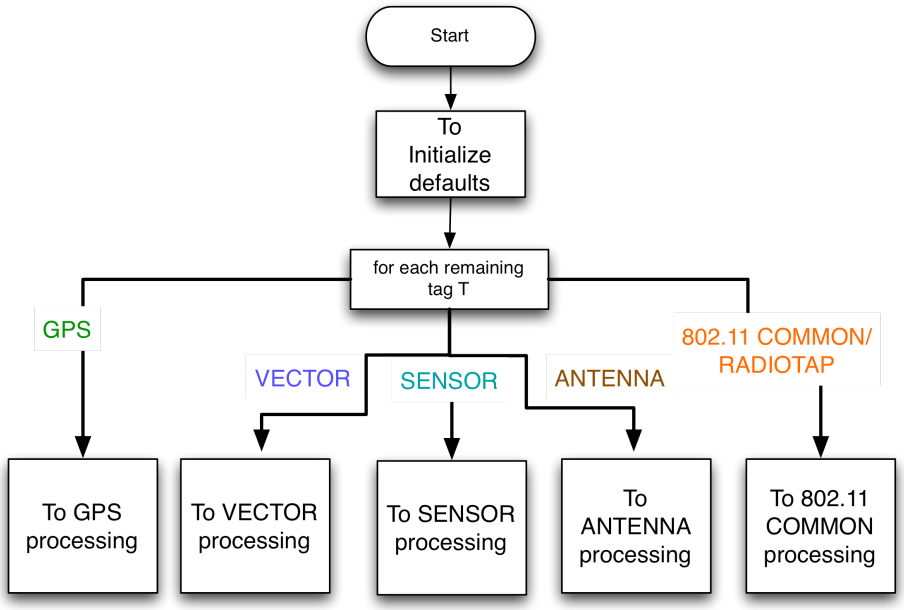
$$= \begin{pmatrix} 0.69\,\text{m West} \\ 0.49\,\text{m North} \\ 0.30\,\text{m Down} \end{pmatrix}$$

P_{ANT_ENU} can be interpreted as follows: when the car is facing East at the given pitch and roll, the antenna is 0.69 meters to the West, 0.49 meters to the North, and 0.3 meters below the center of the vehicle roof. The final absolute location of the antenna on the Earth is the combination of the current GPS latitude, longitude and altitude information with this vector.

9. Processing GEOLOCATION-TAGs

The following section illustrates in detail how GEOLOCATION-TAG processing should be handled. It is not expected that every application will implement this algorithm in its entirety; however applications that deviate from the following interpretation will be viewed as not compliant with the PPI-GEOLOCATION tag standard.

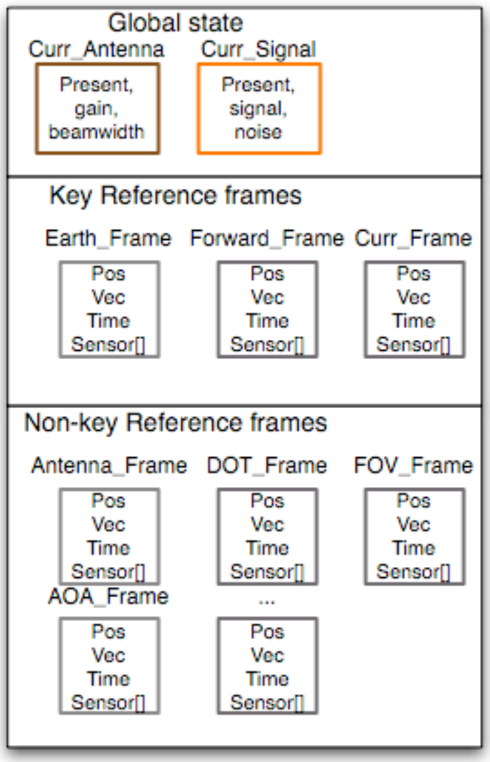
A reference implementation of this algorithm, py-ppi-geo-state, is expected to be available in the PPI-GEOLOCATION SDK. Developers may wish to refer to the code for implementation details. While every effort is made to keep this document and the implementation synchronized, the reference implementation should be seen as the definitive authority on the described state machine.



9.1. *Global GEOLOCATION-TAG state*

The following diagram depicts all of the state that is required to be tracked by a GEOLOCATION-TAG processor. This state includes internal representations of the most recent ANTENNA and 802.11COMMON tag, as well as the three Key Reference frames. A second set of reference frames, ones that correspond to each VectorCharacteristic (Antenna, Direction of Travel, Front of vehicle, etc) is also tracked. These are stored for the convenience of client applications. The state machine does not use these non-key frames of reference as the input of any operation.

It is expected that the internal representation of GEOLOCATION-TAGS preserve the present bitmask to determine whether or not a value has been defined yet. A field that is not present is said to be undefined.



Initalized at start of packet,
Updated upon successful processing of any Antenna or Dot11Common tag, respectively.

Initalized at start of packet,
Reset when GPS tag processed.
Details on when updated below.

Initalized at start of packet
Reset when GPS tag processed.

Updated when any vector tag with the appropriate Vector Characteristic bit is successfully processed,

9.2. *Reference Frames*

Vector tags are combined with GPS tags to define Reference frames. The following rules apply to **all** reference frame variables maintained by the GEOLOCATION-TAG processor.

- All reference frames are reset to their default values at the start of each packet and upon successful processing of any GPS tag.
- The default values for all frames of reference, excluding the Earth Frame, are Off-X=0, Off-Y=0, Off-Z=0, Rot-X=0, Rot-Y=0, Rot-Z=0. This makes the default axes for all non-Earth frames identical to the Earth frame.

9.2.1. Key reference frames

Earth_Frame: Frame of reference where the Pos value is always equal to the most recently processed GPS tag. The Vector portion of Earth frame is immutable, and always represents the E/N/U system described in section 8.3.1.

Curr_Frame This reference frame is updated with the offsets and rotations of the most recently encountered Vector tag. As such, Curr_Frame contains the most recent orientation and position processed by the GEOLOCATION-TAG processor. Applications can check the VectorCharacteristics of this field to determine what this vector indicates (See 10.6 for an example of this).

By default, Off-X=0, Off-Y=0, Off-Z=0, Rot-X=0, Rot-Y=0, Rot-Z=0. This makes all reference frame axes identical to the Earth frame of reference until they are updated.

Forward_Frame: This reference frame is updated with the offsets and rotations of the most recently encountered Vector tag that has the VectorFlags.DEFINES_FORWARD bit set.

This frame of reference is designed to allow applications to create their own convenient local R/F/U based coordinate system. It is described in 8.3.2 and an example of using it is provided in 10.4.

9.2.2. Non-Key reference frames

The GEOLOCATION-TAG processor tracks the following reference frames. They are all updated when the appropriate bit in VectorCharacteristics is set. These are tracked as a convenience for end-user applications; the state machine itself will not use these values as input into any operation.

Antenna_Frame: Indicates the orientation of the most recently processed vector with the VectorChars.ANTENNA bit set.

DOT_Frame: This value contains the most recent frame of reference indicated by a VECTOR with the VectorChars.DIR_OF_TRAVEL bit set.

FOV_Frame: Indicates the orientation of the most recently processed vector with the VectorChars.FRONT_OF_VEHICLE bit set

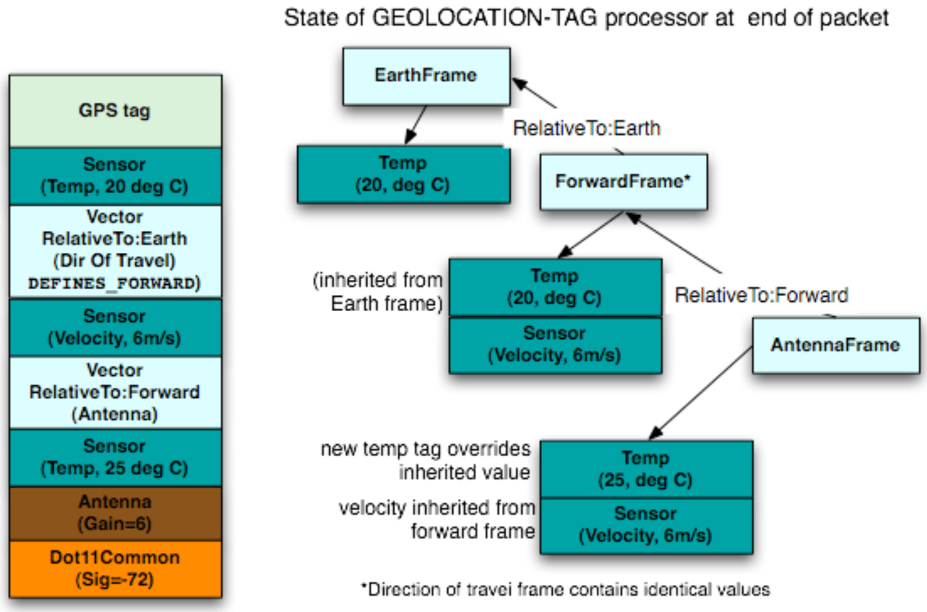
AOA_Frame: Indicates the orientation of the most recently processed vector with the VectorChars.ANGLE_OF_ARRIVAL bit set

Transmitter_POS: Indicates the position of the most recently processed vector with the VectorChars.TRANSMITTER_POS bit set (See example 10.10 for details).

9.3. Sensor Data

An arbitrary amount of sensor data can be attached to any reference frame. This data is stored in a tree-based hierarchy. This hierarchy of this tree is designed to propagate sensor information in a way that is relevant (and well defined) to applications.

Consider the case of a velocity tag that is stored with the Forward frame of reference. In this case a Vector tag that represents an antenna is also provided. The antenna frame is marked as RelativeTo:Forward. In all likelihood the antenna is moving at the same velocity as the reference frame it is RelativeTo. The following algorithm, implemented in the PPI-GEOLOCATION state machine attempts to propagate this information in a meaningful manner, while still giving applications as much flexibility as necessary to store a variety of sensor data.


State of GEOLOCATION-TAG processor at end of packet
GPS tag
Sensor (Temp, 20 deg C)
Vector RelativeTo:Earth (Dir Of Travel) DEFINES_FORWARD)
Sensor (Velocity, 6m/s)
Vector RelativeTo:Forward (Antenna)
Sensor (Temp, 25 deg C)
Antenna (Gain=6)
Dot11Common (Sig=-72)
EarthFrame
Temp (20, deg C)
RelativeTo:Earth
ForwardFrame*
(inherited from Earth frame)
Temp (20, deg C)
Sensor (Velocity, 6m/s)
RelativeTo:Forward
AntennaFrame
new temp tag overrides inherited value
velocity inherited from forward frame
Temp (25, deg C)
Sensor (Velocity, 6m/s)
*Direction of travel frame contains identical values

The following algorithm, implemented inside the PPI-GEOLOCATION state machine attempts to bridge this gap, in a manner that is convenient to both GEOLOCATION-TAG producers and consumers.

- Sensor data that precedes **all** Vector tags is attached to the Earth frame of reference.
- Sensor data which **follows any** Vector tag is attached to all frames of reference that are updated by the most recently preceding Vector tag, (This means the smallest possible subset of updated frames is CurrentFrame).
- Any vector which is RelativeTo: a key frame of reference inherits all sensor data from the key frame it is RelativeTo.

Readers implementing their own PPI-GEOLOCATION tag processor are encouraged to review the reference implementation and ensure equivalent parse trees.

9.4. *GEOLOCATION-TAG state machine variables*

All PPI-GEOLOCATION tag processors must keep track of whether a field in a given variable was initialized to its default value, or if it represents data actually provided by a tag. Fields that are backed with user data are said to be *defined*. Fields which are initialized with values from the state machine are *un-defined*. This relationship is straightforward, except in the case of vectors which are computed RelativeTo: key frames of reference which contained undefined values. In this case the rules in section 9.6.1 are applied.

Unless stated otherwise, all values are set to 0 and undefined at initialization time.
The following section iterates over each variable, and describes when it is initialized and updated.

9.4.1. Curr_Signal

Internal representation of the most recently processed Dot11Common tag. This is where applications look to see the signal strength, channel, etc.

9.4.1.1. Initial values:

Initialized on: PACKET_START
AntSignal, AntNoise are initialized to -128 and undefined (This is used to indicate unknown in Dot11Common tags)
All other Curr_Signal fields are initialized to zero and set to undefined.

9.4.1.2. Updated on:

Successful processing of any Dot11Common-tag causes all fields in Curr_Signal to be updated. Applications that parse Radiotap headers would update all Curr_Signal fields upon successfully parsing of Radiotap headers.

9.4.2. Curr_Antenna

Internal representation of the most recently processed ANTENNA-TAG. This is where applications look to see the beamwidth, gain, etc of the antenna the packet was received on.

9.4.2.1. Initial values:

Initialized on: PACKET_START
Gain is initialized to 5 dBi,
HorizBw is initialized to 360.0
All other Curr_Antenna fields are initialized to zero and set to undefined.
(The motivation for default gain/beamwidths is to provide consistent defaults across various implementations when no antenna data is present.)

9.4.2.2. Updated on:

Successful processing of any ANTENNA-tag causes all fields in Curr_Antenna to be updated.

9.4.3. Reference frames:

Initialized on: PACKET_START, GPS_TAG
Represents various frames of reference in the state machine.

9.4.3.1. Initial values:

Initialized on: PACKET_START, GPS_TAG.
All fields set to zero and undefined on PACKET_START.

Processing of GPS TAGs cause the Position field of **all** reference frames to be updated, and the vector field of all frames other than Earth reset to zero and undefined.

9.4.3.2. Updated on:

Curr_Frame:

Position is updated on processing of any vector (ApplyOffsets).
Orientation is updated on processing of any vector (ApplyRotations).
Sensor data: handled according to 9.3.

Forward_Frame:

Position is updated on processing of any vector with `DEFINES_FORWARD` bit in VectorCharacteristics set (ApplyOffsets).
Orientation is updated on processing of any vector (ApplyRotations).
Sensor data: handled according to 9.3.

Earth_Frame:

Position is updated on processing of any GPS tag.
Orientation is immutable, set to E/N/U coordinate system.
Sensor data: handled according to 9.3.

Non-key-Frames (Antenna, DOT, FOV, …):

Non-key reference frames are updated any time a vector is processed with the applicable VectorCharacteristic bit set.

Position is updated on processing of any applicable vector (ApplyOffsets).
Orientation is updated on processing of any applicable vector (ApplyRotations).
Sensor data: handled according to 9.3.

9.5. *PPI-GEOLOCATION TAG processing overview*

While section 9.4 precisely describes the circumstances when each PPI-GEOLOCATION variable is modified, this section gives a brief description of what happens to a PPI-GEOLOCATION tag is processed. This description is supplementary to the definitions in 9.4

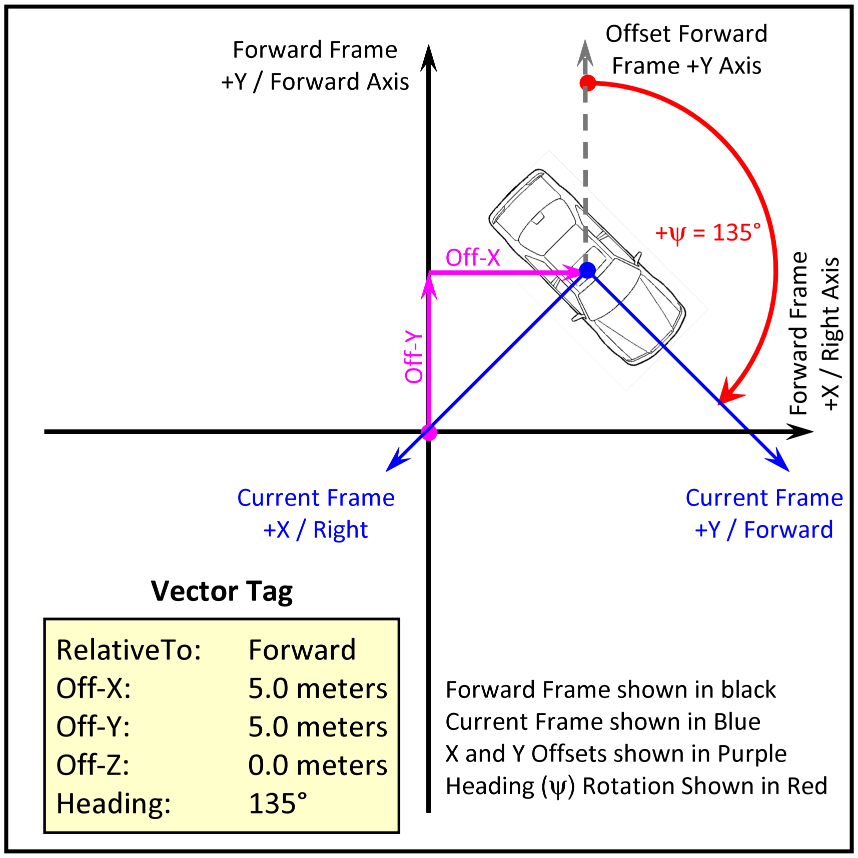
9.5.1. GPS-TAG processing

When A GPS-TAG is processed it causes every frame of reference to take on the supplied value in the position field. They also reset every frame of references vector to their default values.

9.5.2. VECTOR-TAG processing

VECTOR-TAGs are used to precisely define a location (via on offset) and/or an orientation (via the rotation fields). As shown in the Figure below, when both offsets and rotations are present, offsets are applied first. Offset X,Y, and Z values are aligned to the axes of whichever key frame of reference the Vector is RelativeTo. This offset point becomes the origin of the new Current Frame, and any rotations are then applied in the order Heading, Pitch, Roll, centered at this origin. Detailed information on applying rotations is given in Section 8.6.1

Applying Offsets and Rotations


Forward Frame +Y / Forward Axis
Offset Forward Frame +Y Axis
+ψ = 135°
Off-X
Off-Y
Forward Frame +X / Right Axis
Current Frame +X / Right
Current Frame +Y / Forward
Forward Frame shown in black
Current Frame shown in Blue
X and Y Offsets shown in Purple
Heading (ψ) Rotation Shown in Red

Vector Tag

RelativeTo:	Forward
Off-X:	5.0 meters
Off-Y:	5.0 meters
Off-Z:	0.0 meters
Heading:	135°

9.5.3. SENSOR-TAG processing

When a SENSOR-TAG is processed it is attached to one or more reference frames according to the algorithm described in 9.3

9.5.4. ANTENNA-TAG processing

Processing an ANTENNA-TAG causes Curr_Antenna to be Updated. There are no side-effects, and no other variables are modified.

9.5.5. DOT11COMMON-TAG processing

Processing a Dot11Common TAG causes Curr_Signal to be Updated. There are no side-effects, and no other variables are modified.

9.5.6. Invalid tags

It is possible to craft invalid GEOLOCATION-TAGS; either through invalid lengths, invalid fixed point values, and other avenues. Applications encountering any parsing error (other than referencing an undefined value, covered above) should stop processing the current tag, revert GlobalState to the values stored previous to encountering the erroneous tag, and continue processing on the next tag.

9.6. Undefined values

PPI-GEOLOCATION state machines must be able to distinguish between values that were provided as initial defaults, and values that are actually derived from user data. Values that are derived from user data are said to be **defined**. In most cases there is a simple relationship between variables in the PPI-GEOLOCATION state machine.

For example, Curr_Antenna.HorizBw is given an initial value of 360, but it set to undefined As soon as an antenna tag is processed with the HorizBw field, the variable will be updated and marked as defined. This simple relationship holds true for all variables except the vector portion of reference frames. Since vectors can be computed relative to other vectors, and any one of the 6 rotation variables involved can be undefined, the following rules are used.

9.6.1. Definedness of ReferenceFrame.Vec

1. Any rotations not present will be assumed 0 and marked undefined in the resulting Reference Frame.

2. If two vectors are combined that BOTH provide ONLY Heading, ONLY Pitch, or ONLY Roll, then the provided Heading, Pitch or Roll will be defined, and the other two rotations will be assumed 0 and marked undefined in the resulting Reference Frame.

3. Otherwise, if two vectors that both have at least one rotation present are combined, then if any Pitch, Roll, Heading value for either of the two vectors was undefined, all three will be undefined in the resulting Reference Frame.

The application of rule 2 allows many applications only concerned with heading to get the desired results from the state machine, while rule 3 allows applications to validate that all the inputs were present to reliably compute the vector used in the reference frame.

9.6.2. Undefined altitudes

Applications encountering packets with no altitude or altitude_g field must assume they were captured at ground level. More precisely: when a frame of reference has an undefined Pos.altitude and an undefined Pos.altitude_g field, applications shall treat the altitude as ground level.

10. Applications

The most obvious use for GEOLOCATION-TAGS is to provide a standardized way for 802.11 capturing utilities to encode GPS and orientation information. Assuming a fully filled out set of GEOLOCATION-TAGS, the visualization system can know exactly when and where a packet was captured, including the orientation of the antenna and the antenna in use. Supplementary sensor information can also be provided, which allows the precise expression of velocity, acceleration, temperature, and can easily be expanded to include application specific data. It is the author's hope that by commoditizing the storage of this information interest will develop around building geo-location systems that can interoperate with common tools.

Common use cases

The following section identifies the most common use-cases for GEOLOCATION-TAGS. It starts out with the simplest cases to encode and moves on to more difficult examples. Although not every conceivable combination is expressed below, the most useful cases should be covered. Applications are encouraged to imitate one of these use-cases in order to maximize compatibility. A pcap file that illustrates all of these examples is included in the SDK.

10.1. GPS Only

Many applications today store GPS meta-information out-of-band of packets captured during a survey. These applications could be easily modified to output this information in-line with the pcap file by using a GPS-TAG. The following scenario walks through the state of a GEOLOCATION-TAG processor that encounters such input.

GPS-TAG	
Latitiude	40.787743°
Longitude	-73.971210°

Processing this single tag results in the following GEOLOCATION-STATE table.
A (**u**) next to a name indicates that the current value is undefined, a value provided by or derived from a default.

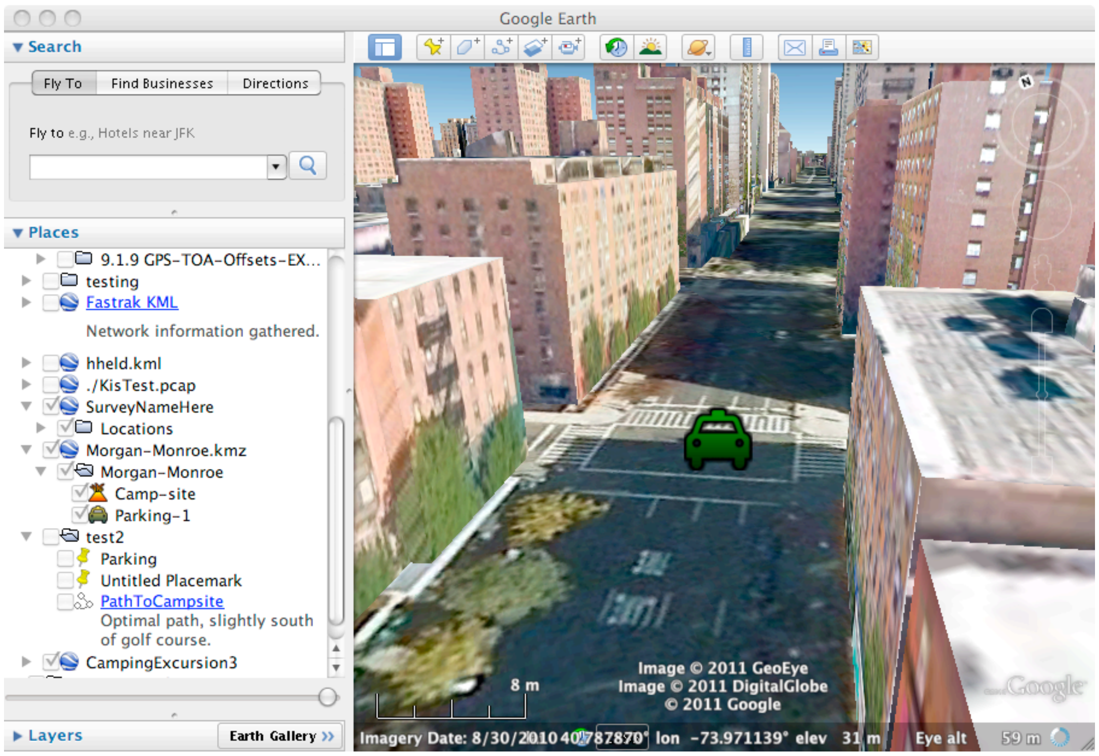
Antenna_Frame	
Position	
Defined:	Lon,Lat
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	None
Rot-X (Pitch) (**u**)	0
Rot-Y (Roll) (**u**)	0
Rot-Z (Heading) (**u**)	0
Curr_Antenna (defaults)	
Defined:	None
AntennaFlags (**u**)	0x00
Gain (**u**)	5
Horizbw (**u**)	360.0

Curr_Sig (undefined)	
Defined:	None
Antsignal (**u**)	-128 (unknown, from initialize_defaults)
Antnoise (**u**)	-128 (unknown, from initialize_defaults)

Curr_Frame and ForwardFrame are identical to Antenna_Frame

Since no ANTENNA-TAG was included, the defaults specified in 9.4.2.1 were used to populate Curr_Antenna.

The preceding GEOLOCATION-STATE indicate that there is an omni-directional antenna pointed due north (AntennaFrame.rotZ=0) located at the position described by the GPS-TAG. However, the rotation information is marked as undefined (since it did not come from any user data). Antenna characteristics as well as signal strength are also marked as undefined in the state machine. The visualization output shown below is based only on variables in the state machine that are marked as defined.



10.2. GPS + VECTOR + ANTENNA + RADIOTAP

This case represents a common scenario where the collection system has a single omni-directional antenna on the top of the vehicle.

GPS-TAG	
Latitude	40.787743°
Longitude	-73.971210°

VECTOR (Antenna)	
VectorFlags	0x02 RelativeTo:Earth
VectorChars	0x01 (Antenna)
Rot-X: (Pitch)	90
Rot-Y: (Roll)	0
Rot-Z: (Heading)	0
DescriptionString	Antenna-1 orientation

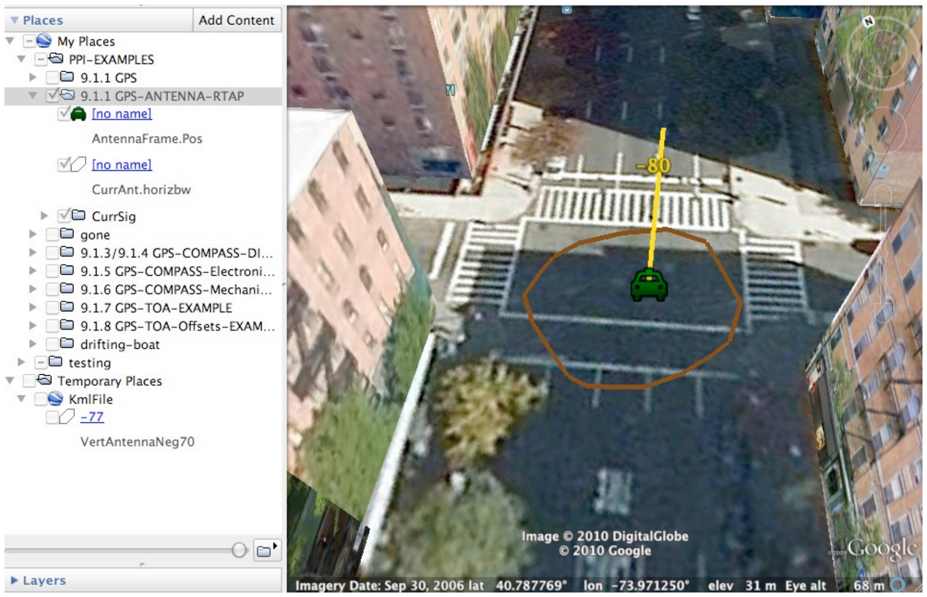
ANTENNA-TAG	
AntennaFlags	0x02 Bit 1: Horizontal-Polarity
Gain	8 dBi
HorizBW	360.0
Modelname	8dBi-MagMountOmni

RADIOTAP	
Antsignal	-80 dB
Antnoise	-110 dB
channel	6

The addition of a Vector-Tag, ANTENNA-TAG and Radiotap header results in the following updated GEOLOCATION-STATE

Antenna_Frame	
Position	
Defined:	Lon,Lat
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	Rot-X/Y/Z
Rot-X: (Pitch)	90
Rot-Y: (Roll)	0
Rot-Z: (Heading)	0
Curr_Antenna	
Defined:	AntennaFlags,gain,horizbw,Modelname
AntennaFlags	0x02 Bit 1: HORIZONTALLY_POLARIZED
Gain	8
horizbw	360.0
Modelname	8dBi-MagMountOmni

Curr_Sig	
Defined:	Antsignal,Antnoise,Channel
Antsignal	-80
Antnoise	-110
Channel	6



In the image above the visualization system renders signal strength as a color coded and scaled line, where orientation is determined by AntennaFrame.Vec.

10.3. Directional of travel +Velocity + one directional antenna

In this case we illustrate a common configuration of one directional antenna pointed out the passenger side window of a moving car. This requires the introduction of two VECTOR-TAGs; one for the direction of the car, and one for the antenna. Note that the second VECTOR tag utilizes rotations RelativeTo:ForwardFrame. Also note the introduction of the Rot-X field in the vehicle's VECTOR TAG; the system is now recording a 10 degree incline. This example also introduces the sensor tag.

GPS (vehicle position)	
GPSFlags	0x02 1:GPS_FIX
Latitude	40.787743°
Longitude	-73.971210°

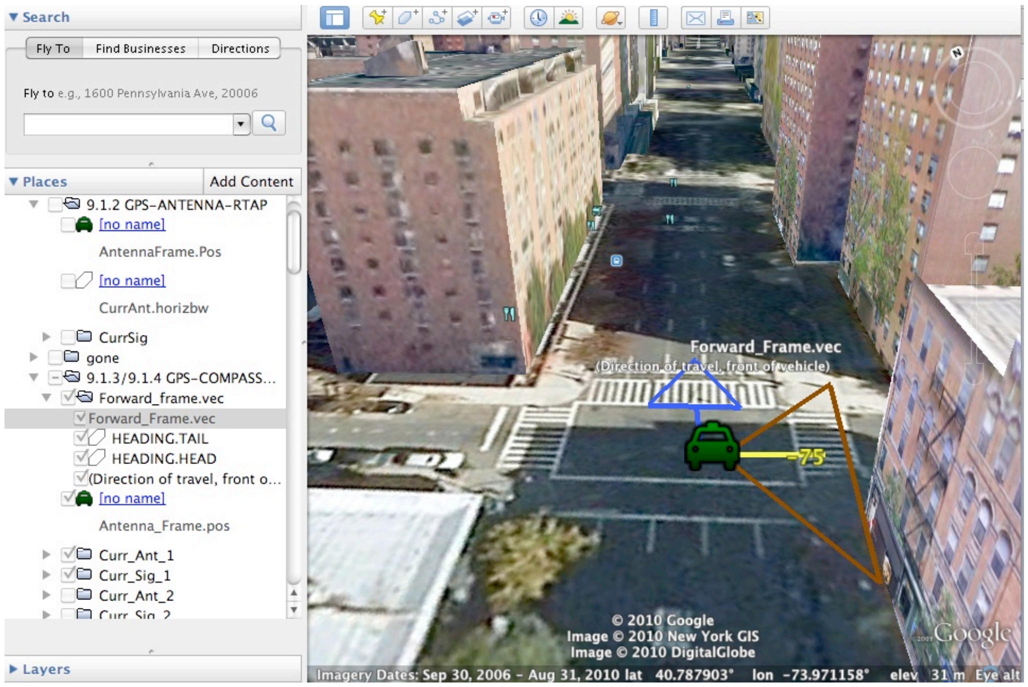
VECTOR (vehicle orientation)	
VectorFlags	0x03 Defines_Forward RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x06 1:DIRECTION_OF_TRAVEL 2:FRONT_OF_VEHICLE
Rot-X: pitch	10.0°
Rot-Z: heading	22.5° (NNE)

Sensor (Velocity)	
SensorType	Velocity
Val_T	20.0 m/s

VECTOR (right antenna orientation)	
VectorFlags	0x04 RelativeTo:ForwardFrame
VectorChars	0x01 Bit 0:ANTENNA
Rot-Z: heading	90.0 (90 degrees right, relative to forward)

ANTENNA (right antenna)	
Antennaflags	0x02 Bit 1: HORIZONTALLY_POLARIZED
Gain	9 dBi
horizbw	120
Modelname	SA24-120-9

802.11COMMON1 (right antenna signal)	
Antsignal	-75 dB
Antnoise	-110 dBi
channel	6



The GEOLOCATION-STATE generated from processing the preceding five tags is shown below.

Forward_Frame*	
Position	
Defined:	GPSFlags,Lon,Lat
GpsFlags	0x02: GPS-FIX
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	VectorFlags,VectorChars,RotX,RotZ
VectorCharacteristics	0x06 FRONT_OF_VEHICLE DIRECTION_OF_TRAVEL
Rot-X: Pitch	10.0°
Rot-Y: Roll (**u**)	0° (comes from initialize_defaults)
Rot-Z: Heading	22.5.° (NNE)
SensorData (Velocity)	
Val-T	20.0 m/s

*DOT_Frame, FOV_Frame contain identical values.

Antenna_Frame	
Position	
Defined:	Lon,Lat
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	VectorFlags, VectorChars, RotX ,RotZ,
VectorCharacteristics	0x01 Bit 0: ANTENNA
Rot-X: Pitch (**u**)	0.0
Rot-Y: Roll (**u**)	10.0

Rot-Z: Heading (**u**)	112.5° (ESE)
SensorData (Velocity)	
Val-T	20.0 m/s (Inherited from Forward_ Frame)

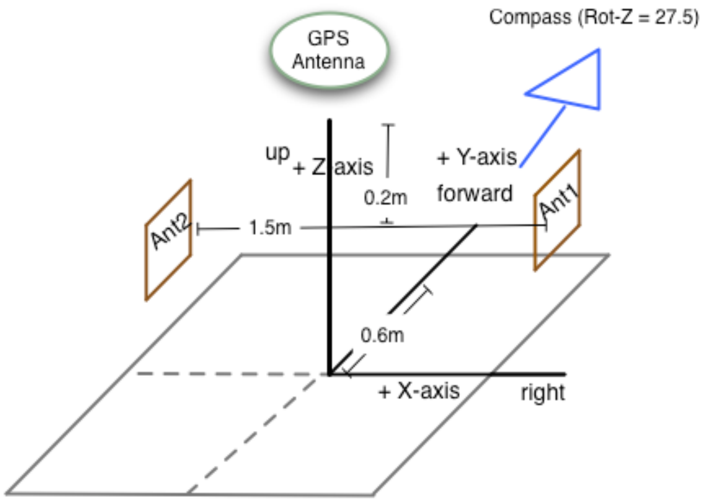
Examining AntennaFrame.Vec field we see that 10° of pitch relative to the vehicle is equal to 10° of roll to the antenna. This is consistent with an antenna pointed out the passenger window of an up-hill travelling car. Notice that since Roll was not included in the vector tag that created ForwardFrame, roll was not defined in ForwardFrame.Vec. Therefore when the state machine combined the antenna vector with ForwardFrame according to 9.6.1 it marked all of the Vec fields in AntennaFrame as undefined.

We can follow the propagation of the velocity Sensor tag throughout the frames of reference in this example. The tag was originally attached to the ForwardFrame, and since the second vector tag was RelativeTo:Forward, the sensor data propagated to AntennaFrame as well.

10.4. Two static directional antennas with offsets

This case is similar to the previous, except that we now have two antennas connected to two independent interfaces capturing the same packet. (How an application verifies that this was indeed the same packet before tagging it in this manner is outside the scope of this document.).

Assume for simplicity that the vehicle is exactly 1.5 meters wide, that the GPS Antenna is located 0.2 meters above each antenna (on the roof), and that each antenna is forward 0.6m of the GPS antenna. If we wanted to encode all of this precisely, we would do it as follows.



GPS (vehicle position)	
GPSFlags	0x02 1:GPS_FIX
Latitiude	40.787743°
Longitude	-73.971210°
Altitude_g	2.0m

Note the use of altitude_g here: We specify the height of the roof of the car as 2.0m from ground. **It is recommended that ground based applications utilize the altitude_g field** rather filling in altitude results from a GPS receiver, as altitude readings may vary significantly in values and precision in real world conditions. Alternately, applications may simply leave altitude out, which should cause all applications processing geolocation-tags to assume ground level.

VECTOR (vehicle orientation)	
VectorFlags	0x03 Defines_Forward RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x06 1:DIRECTION_OF_TRAVEL 2:FRONT_OF_VEHICLE
Rot-X: pitch	10.0°
Rot-Z: heading	22.5° (NNE)

Sensor (Velocity)	
SensorType	Velocity
Val_T	8.5 m/s

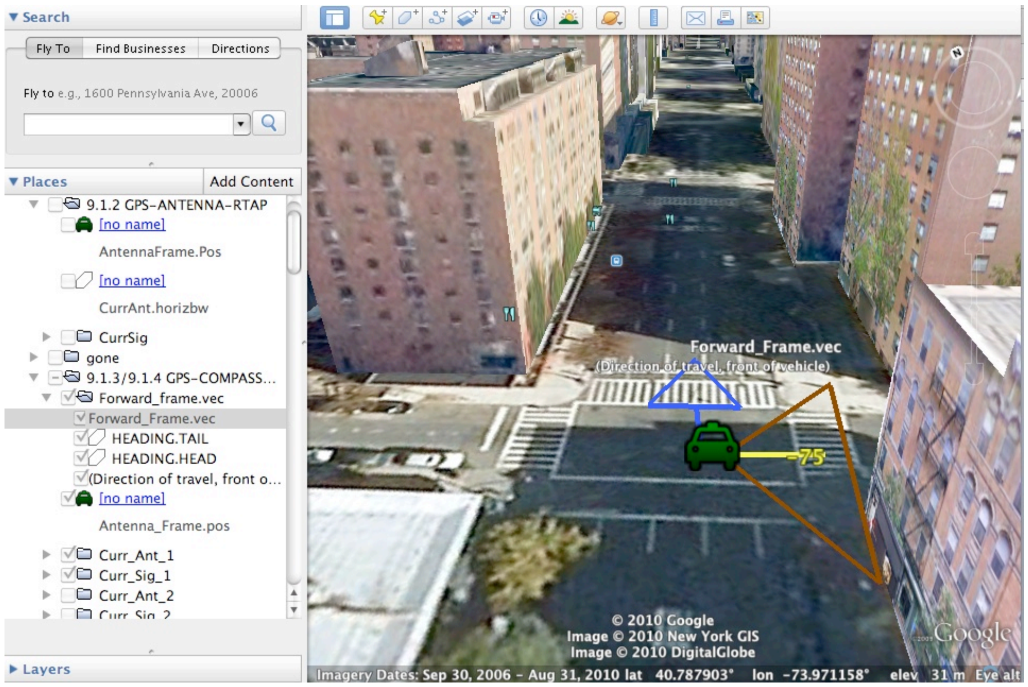
Sensor (Acceleration)	
SensorType	Acceleration
Val_T	0.5 m/s^2

The velocity and acceleration values tell us that the car was travelling through the intersection when this packet was captured.

VECTOR (right antenna orientation)	
VectorFlags	0x04 RelativeTo:ForwardFrame
VectorChars	0x01 Bit 0:ANTENNA
Rot-Z: heading	90.0 (90 degrees right, relative to forward)
Off-X: (Right)	0.75m
Off-Y: (Forward)	0.6m
Off-Z: (Up)	-0.2m

ANTENNA (right antenna)	
Antennaflags	0x02 Bit 1: HORIZONTALLY_POLARIZED
Gain	9 dBi
horizbw	120
Modelname	SA24-120-9

802.11COMMON1 (right antenna signal)	
Antsignal	-75 dB
Antnoise	-110 dBi
channel	6



Lets examine the values of the GEOLOCATION-STATE at this point.

Forward_Frame	
Position	
Defined:	GPSFlags,Lon,Lat,Alt_g
GpsFlags	0x02: GPS-FIX
Lat/Lon/Alt_**g**:	40.787743°, -73.971210°, 2.0m
Vec	
Defined:	VecFlags,VectorChars,RotX,RotZ
VectorCharacteristics	0x06 FRONT_OF_VEHICLE DIRECTION_OF_TRAVEL
Rot-X: Pitch	10.0°
Rot-Y: Roll (**u**)	0° (This is a default)
Rot-Z: Heading	22.5.° (NNE)
SensorData (Velocity)	
Val-T	8.5 m/s
SensorData (Acceleration)	
Val-T	0.5 m/s^2

Forward_Frame was defined by the combination of the first GPS-TAG and the first VECTOR-TAG. We also have acceleration and velocity information from the sensor tags.

Antenna_Frame's position has an offset of (.75m right, .6m forward, 0.2m down) of the position stored in Forward_Frame.pos. Converting these offsets into the E/N/U coordinate system (as shown in 8.6.4) we arrive at: (0.93m East, 0.29m North, -0.09 Up). Applying the E/N/U offsets to the original lon/lat/alt leads to the results listed in Anntenna_Frame.pos

Antenna_Frame, CurrFrame	
Position	
Defined:	Lon,Lat,Alt_g
Lat/Lon/Alt_**g**:	40.7877459, -73.9711987, 1.8m
Vec	
Defined:	VectorChars,RotX RotZ,
VectorCharacteristics	0x06 FRONT_OF_VEHICLE DIRECTION_OF_TRAVEL
Rot-X: Pitch (**u**)	0.0
Rot-Y: Roll (**u**)	10.0
Rot-Z: Heading (**u**)	115.5° (ESE)
SensorData (Velocity)	(Inherited from ForwardFrame)
Val-T	8.5 M/s
SensorData (Accel)	(Inherited from ForwardFrame)
Val-T	0.5 M/s^2

Examining AntennaFrame.Vec field we see that 10° of pitch relative to the vehicle is equal to 10° of roll to the antenna. This is consistent with an antenna pointed out the passenger window of an up-hill travelling car. We can see that all of the rotation fields on Antenna frame are undefined, due to rule 3 of section 9.6.1.

Curr_Antenna and Curr_Signal are trivially derived from their respective tags.

Curr_Antenna	
Defined:	AntennaFlags,gain,horizbw,Modelname
Undefined:	Vertbw,PrecisionGain,BeamID, ...
AntennaFlags	0x02 Bit 1: HORIZONTALLY_POLARIZED
gain	9
horizbw	120.0
Modelname	SA24-120-9
Curr_Sig	
Defined:	Antsignal, Antnoise, channel
Undefined:	…
Antsignal	-75
Antnoise	-110
channel	6

If we were to plot out all of the information present so far, it would look similar to the illustration in 10.4

Moving on to the next set of tags we have:

VECTOR (left antenna orientation)	
VectorFlags	0x00 RelativeTo:ForwardFrame
VectorChars	0x01 Bit 0:ANTENNA
Rot-Z: heading	270.0 (degrees right, relative to forward)
Off-X: (Right)	-0.75m
Off-Y: (Forward)	0.6m
Off-Z: (Up)	-0.2m

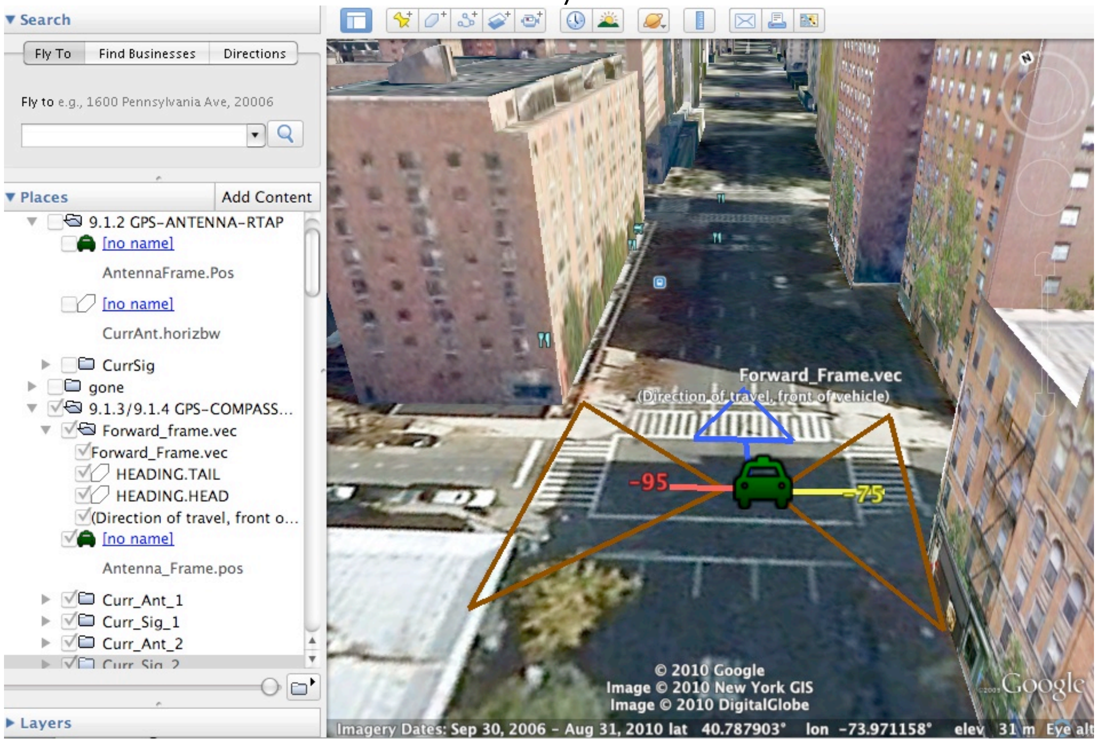
ANTENNA (left antenna)	
Antennaflags	0x02 Bit 1: HORIZONTALLY_POLARIZED
Gain	9 dBi
horizbw	120
Modelname	SA24-120-9

802.11COMMON1 (left antenna signal)	
Signal	-95 dB
noise	-118 dBi
channel	6

The VECTOR-TAG changes both Curr_Frame and Antenna_frame to reflect the offsets and rotations indicated. Instead of slightly north-east of our initial point, we are slightly north-west. And since this antenna is 180 degrees opposite the first, the 10 degrees of roll are in the opposite direction.

Antenna_Frame, CurrFrame	
Position	
Defined:	GPSFlags, Lon, Lat, Alt_g
Lat/Lon/Alt_**g**:	(-0.75R, 0.60F, 0.20U)-->(-0.45E, 0.87N,-0.09U) 40.7877521, -73.9712145, 1.8M
Vec	
Defined:	VectorFlags,VectorChars, RotX RotZ, Off-R/F/U
VectorCharacteristics	0x06 FRONT_OF_VEHICLE DIRECTION_OF_TRAVEL
Rot-X: Pitch(**u**)	0.0°
Rot-Y: Roll (**u**)	-10.0°
Rot-Z: Heading (**u**)	292.5° (WNW)
SensorData (Velocity)	(Inherited from ForwardFrame)
Val-T	8.5 m/s
SensorData (Accel)	(Inherited from ForwardFrame)
Val-T	0.5 m/s^2

Of course Curr_Sig and Curr_Antenna are updated accordingly, resulting in the following diagram. (Signal strength has been colored/scaled to reflect intensity), and the offsets were of such a small scale they are difficult to see.


Search
Fly To
Find Businesses
Directions
Fly to e.g., 1600 Pennsylvania Ave, 20006
Places
Add Content
Layers
Forward_Frame.vec
(Direction of travel, front of vehicle)
-95
-75
© 2010 Google
Image © 2010 New York GIS
Image © 2010 DigitalGlobe
Google

9.1.2 GPS-ANTENNA-RTAP
[no name]
AntennaFrame.Pos
[no name]
CurrAnt.horizbw
CurrSig
gone
9.1.3/9.1.4 GPS-COMPASS...
Forward_frame.vec
Forward_Frame.vec
HEADING.TAIL
HEADING.HEAD
(Direction of travel, front o...
[no name]
Antenna_Frame.pos
Curr_Ant_1
Curr_Sig_1
Curr_Ant_2
Curr_Sig_2

Imagery Dates: Sep 30, 2006 – Aug 31, 2010 lat 40.787903° lon -73.971158° elev 31 m Eye alt

The above screenshot conveys most of this information, although seeing the offsets and pitch is difficult due to scale.

10.5. Electronically steerable antenna

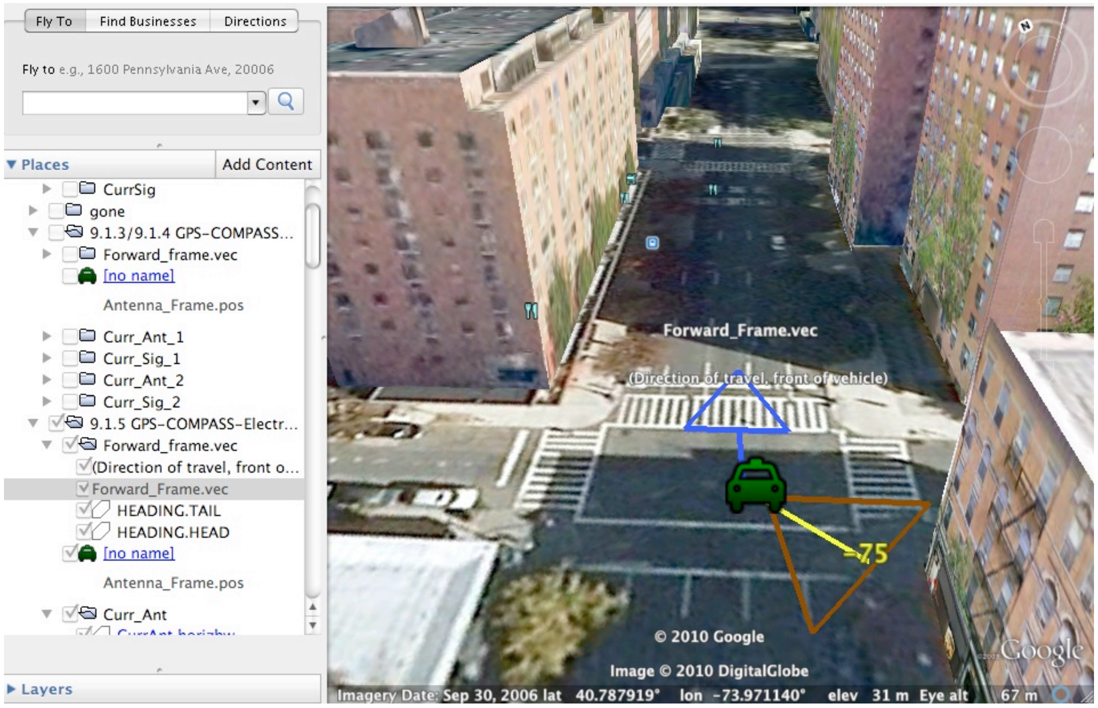
This case is similar to that of 10.2, except that the ELECTRONICALLY_STEERABLE_ ANTENNA bit in the AntennaFlags is set. Antennas of this type can electronically steer their beam, resulting in many packets captured at approximately the same time/location/ but varying rot-Z values for the antenna. In this case we tagged a packet while the antenna was electronically steered 120 degrees from the front. (Astute readers will note the presence of the BeamID field in the ANTENNA-TAG. This 16-bit value should uniquely identify every possible beam pattern present for a given antenna.)

GPS (vehicle position)	
GPSFlags	0x02 Bit 1:GPS_FIX
Latitiude	40.787743°
Longitude	-73.971210°

VECTOR (current orientation of electrically-steered antenna)	
VectorFlags	0x03 Defines_Forward RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x01 Bit 0:ANTENNA
Rot-Z: heading	120.0 (degrees right, relative to forward)

ANTENNA (Electronically steerable)	
Antennaflags	0x00010002 Bit 2: HORIZONTALLY_POLARIZED Bit 16: ELECTRONICALLY_STEERABLE
Gain	12 dBi
horizbw	60
BeamID	0xF1A1 (Vendor specific)
Modelname	ElectronicallySteerableExAntenna
AppId	0x04030201

802.11COMMON1 (electronically steerable antenna signal)	
Signal	-75 dB
noise	-110 dBi
channel	6


Fly To
Find Businesses
Directions
Fly to e.g., 1600 Pennsylvania Ave, 20006
Places
Add Content
CurrSig
gone
9.1.3/9.1.4 GPS-COMPASS...
Forward_frame.vec
[no name]
Antenna_Frame.pos
Curr_Ant_1
Curr_Sig_1
Curr_Ant_2
Curr_Sig_2
9.1.5 GPS-COMPASS-Electr...
Forward_frame.vec
(Direction of travel, front o...
Forward_Frame.vec
HEADING.TAIL
HEADING.HEAD
[no name]
Antenna_Frame.pos
Curr_Ant
Layers
Forward_Frame.vec
(Direction of travel, front of vehicle)
-75
© 2010 Google
Image © 2010 DigitalGlobe
Google
Imagery Date: Sep 30, 2006 lat 40.787919° lon -73.971140° elev 31 m Eye alt 67 m

10.6. Mechanically steerable antenna

This case is similar to 10.5 except that it illustrates that the forward frame of reference may differ from the capturing systems "intuitive" notion of forward. This example utilizes three different VECTOR-TAGs. One to define the forward frame of reference, one to define the direction of travel/front of vehicle, and finally a relative vector to describe the orientation of a mechanically steered antenna.

The motivation for this case is a mechanically steered antenna positioned out the rear of a vehicle, thus the servo's definition of forward is out the back of the car. In this case the mechanically steered antenna is 75 degrees past the zero mark.

GPS (vehicle position)	
GPSFlags	0x02 1:GPS_FIX
Latitiude	40.787743°
Longitude	-73.971210°

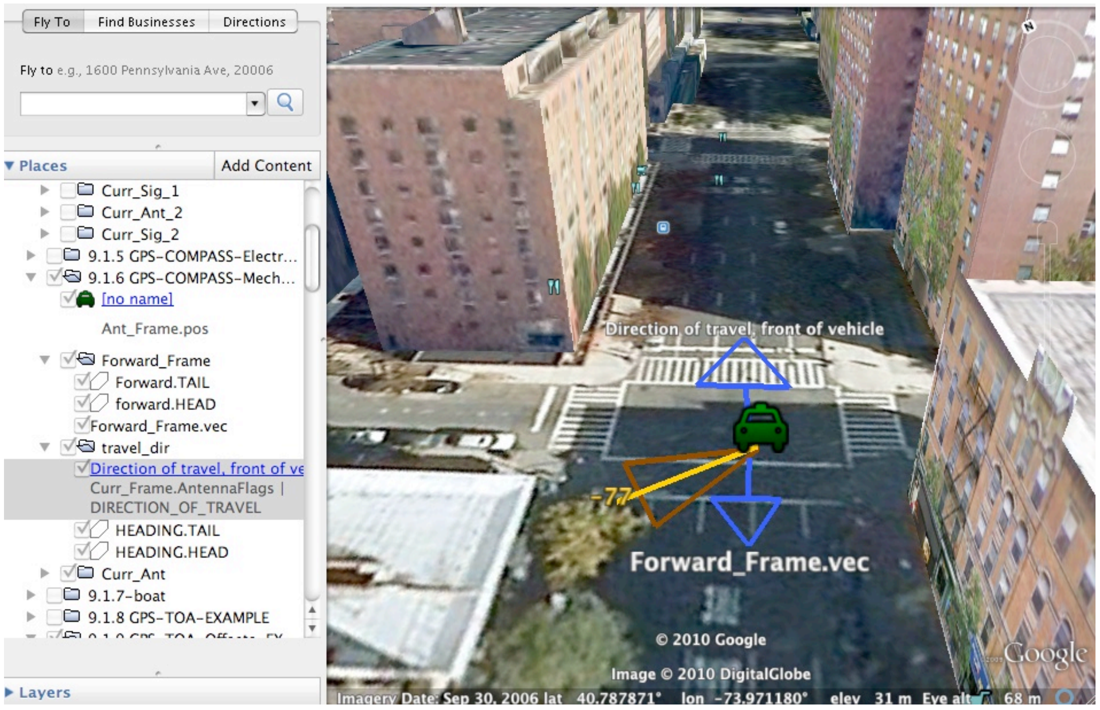
VECTOR (vehicle orientation, direction of travel)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x06 1:DIRECTION_OF_TRAVEL 2:FRONT_OF_VEHICLE
Rot-Z: heading	22.5.(NNE)

VECTOR (forward frame of reference (back of vehicle)	
VectorFlags	0x03 Defines_Forward RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x00
Rot-Z: heading	202.5°(SSW)

VECTOR (mechanically steerable antenna orientation)	
VectorFlags	0x00 RelativeTo:Forward
VectorChars	0x01 1:ANTENNA
Rot-Z: heading	75.0°

ANTENNA (Electronically steerable)	
Antennaflags	0x020002 Bit 2: HORIZONTALLY_POLARIZED Bit 17: ELECTRONICALLY_STEERABLE
Gain	12 dBi
horizbw	60
Modelname	12dBi-Panel

802.11COMMON1 (steerable antenna signal)	
Antsignal	-77 dB
Antnoise	-110 dBi
channel	6


Fly To
Find Businesses
Directions
Fly to e.g., 1600 Pennsylvania Ave, 20006
Places
Add Content
Curr_Sig_1
Curr_Ant_2
Curr_Sig_2
9.1.5 GPS-COMPASS-Electr...
9.1.6 GPS-COMPASS-Mech...
[no name]
Ant_Frame.pos
Forward_Frame
Forward.TAIL
forward.HEAD
Forward_Frame.vec
travel_dir
Direction of travel, front of ve
Curr_Frame.AntennaFlags | DIRECTION_OF_TRAVEL
HEADING.TAIL
HEADING.HEAD
Curr_Ant
9.1.7-boat
9.1.8 GPS-TOA-EXAMPLE
Layers
Direction of travel, front of vehicle
-77
Forward_Frame.vec
© 2010 Google
Image © 2010 DigitalGlobe
Imagery Date: Sep 30, 2006 lat 40.787871° lon -73.971180° elev 31 m Eye alt 68 m
Google

Forward_Frame	
Position	
Defined:	GPSFlags,Lon,Lat
GpsFlags	0x02: GPS-FIX
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	VectorChars,RotZ
Rot-X:Pitch (**u**)	0° (This value comes from initialize_defaults)
Rot-Y:Roll (**u**)	0° (This value comes from initialize_defaults)
Rot-Z: Heading	202.5° (SSW)

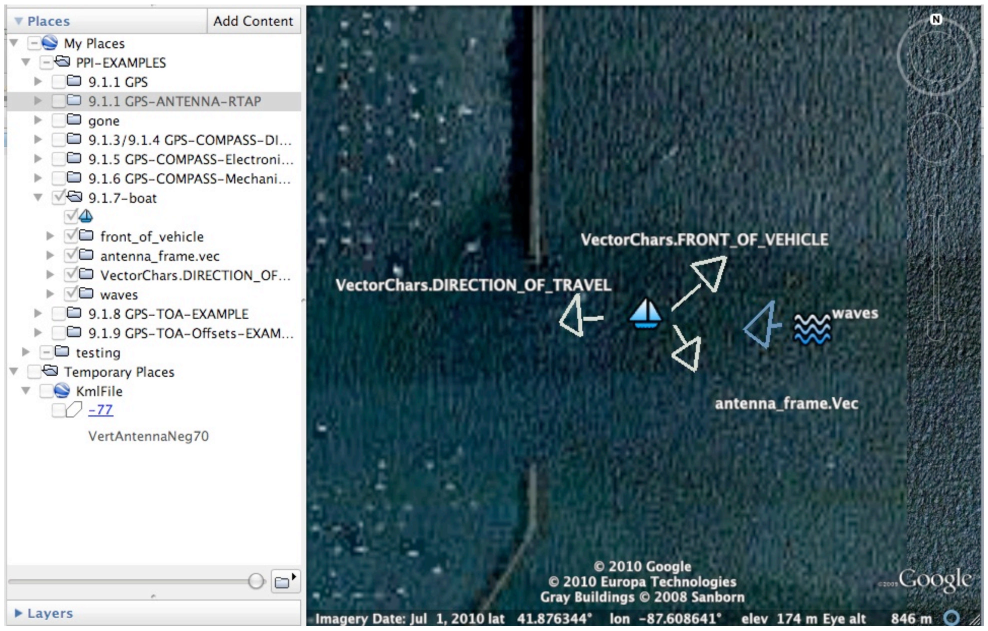
Antenna_Frame, Curr_Frame	
Position	
Defined:	GPSFlags,Lon,Lat
GpsFlags	0x02: GPS-FIX
Lat/Lon	40.787743°, -73.971210°
Vec	
Defined:	VectorFlags,VectorChars,RotZ
VectorChars	0x01 1:ANTENNA
Rot-X: Pitch (**u**)	0° (This value comes from RFU2ENU)
Rot-Y: Roll (**u**)	0°
Rot-Z: Heading	277.5° (WSW)

Note that the heading in AntennaFrame is marked as defined. This is because rule 2 in section 9.6.1 has been applied.

Note that the original vector that indicated direction of travel is contained in DOT_Frame. It is omitted for brevity.

10.7. Drifing boat

This example is included for sake of completeness; Imagine a boat at sea oriented toward the north east, with one antenna off the starboard (right) side, drifting due West. In this case the direction of travel is West, the front of the vehicle is North East, and the antenna orientation is South East. This would be encoded with three different VECTOR tags, each with a specific bit in VectorChars set. This situation would look something like the following.


Places
Add Content
My Places
PPI-EXAMPLES
9.1.1 GPS
9.1.1 GPS-ANTENNA-RTAP
gone
9.1.3/9.1.4 GPS-COMPASS-DI...
9.1.5 GPS-COMPASS-Electroni...
9.1.6 GPS-COMPASS-Mechani...
9.1.7-boat
front_of_vehicle
antenna_frame.vec
VectorChars.DIRECTION_OF...
waves
9.1.8 GPS-TOA-EXAMPLE
9.1.9 GPS-TOA-Offsets-EXAM...
testing
Temporary Places
KmlFile
-77
VertAntennaNeg70
Layers
VectorChars.FRONT_OF_VEHICLE
VectorChars.DIRECTION_OF_TRAVEL
waves
antenna_frame.Vec
© 2010 Google
© 2010 Europa Technologies
Gray Buildings © 2008 Sanborn
Google
Imagery Date: Jul 1, 2010 lat 41.876344° lon -87.608641° elev 174 m Eye alt 846 m

The goal of this example is to illustrate why applications should pay attention to how they parse and set VectorCharacteristics.

10.8. Time of arrival analysis

This example illustrates how to utilize the TDOA-CLOCK Sensor to encode timing and position values with sufficient precision that an angle of arrival can be computed based on time of flight using two antennas, or a position solution can be found using three antennas.

In this case two antennas are positioned 50 meters apart on Soldier field. Each portion of the collection system shares a synchronized clock with nano-second resolution. Since the distance between the antennas is significant, they are marked with GPS-Tags. Applications can encode distances with millimeter precision by utilizing the offset field of Vector-TAGS if necessary.

GPS-TAG (ant1)	
GPS-Flags	0x80 8:Manual input mode
Latitude	41.861885
Longitude	-87.616926
GPSTime	Nov 02 2010 17:58:39
FractionalTime	.20

VECTOR (first antenna)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x01 Bit 0: Antenna
Rot-X: (Pitch)	90.0
DescriptionString	Antenna-1 orientation

SensorData (TDOA_CLOCK)	
ScaleFactor	-9
Val-T	**60.8754** (sec)
AppId	0x04030201

ANTENNA-TAG 1	
Gain	5 dBi
HorizBW	360.0
AntennaFlags	Bit 1: Horizontal-Polarity 1
Modelname	8dBi-Omni
DescriptionString	Signal 1

802.11COMMON1 (ant1 signal)	
Antsignal	-60 dB

This is the first time we have seen a Sensor tag that utilizes the ScaleFactor field. This sensor indicates that the shared clock is set to 10^{-9} *60.8754 secs. (To get an absolute time this value can be added to the GPSTimestamp field in the GPS-TAG; however most applications using TDOA_CLOCK will be only interested in the deltas between them.) The preceding tags define a precise location and time when the packet was received.

The following tags describe the second antenna, where the packet was received 118 ns later.

GPS-TAG (ant2)	
GPS-Flags	0x10 7:Manual input mode
Latitude	41.861904
Longitude	-87.616365
GPSTime	Nov 02 2010 17:58:39
FractionalTime	.20

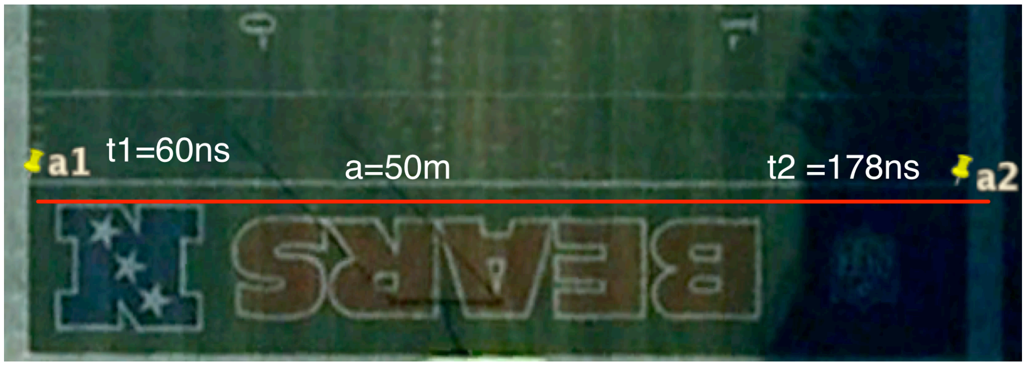
VECTOR (second antenna orientation)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x01 0: Antenna
Rot-X: (Pitch)	90.0
DescriptionString	Antenna-1 orientation

SensorData (TDOA_CLOCK)	
ScaleFactor	-9
Val-T	**178.124** (Secs)
AppId	0x04030201

ANTENNA-TAG 2	
Gain	5 dBi
HorizBW	360.0
AntennaFlags	Bit 1: Horizontal-Polarity 1
Modelname	8dBi-MagMountOmni
DescriptionString	“Bottom Right of field”

802.11COMMON1 (ant1 signal)	
Antsignal	-80 dB

Again, the field to note is the TDOA_Clock which is set to 10^{-9} *178.124 secs

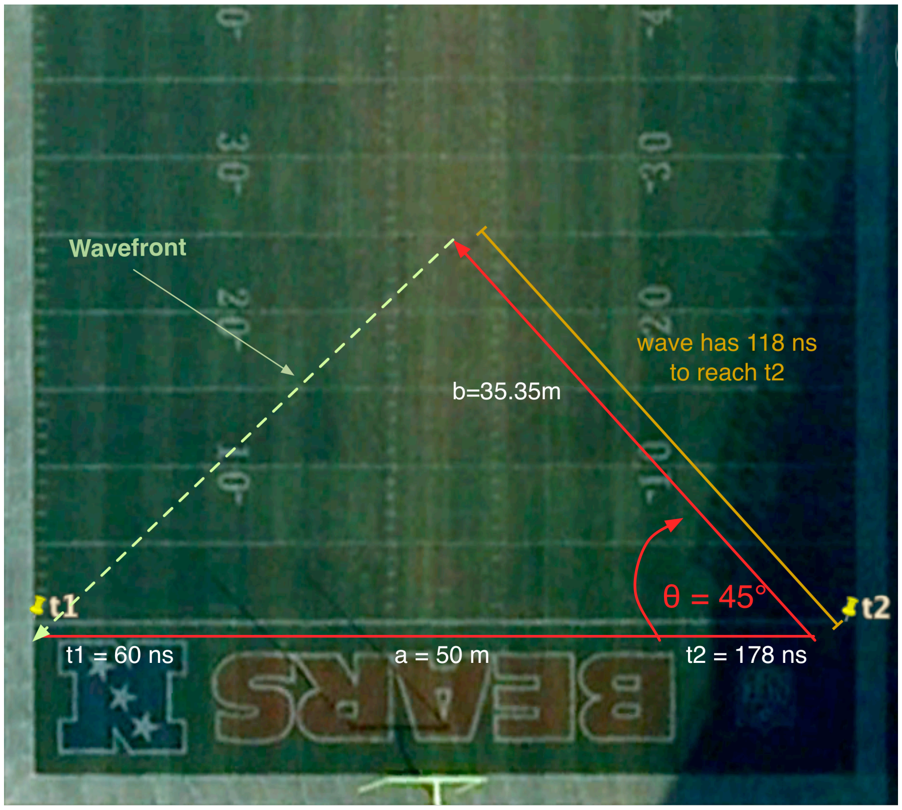


By taking the delta between the two clocks we can determine that when the wave arrived at t1, it took 118 ns to reach t2. Light travels approximately 35 meters in 118ns , which means that when the wave had arrived at t1 it still had 35 meters to travel to t2.

dt = 178 – 60 = 118ns
b = 118ns * (3e8ms)
b = 35.35 meters

Given that we know the distance between both antennas, and given that we know the distance the signal had to travel after encountering t1 to arrive at t2, we can perform the following calculation to compute the angle of arrival at t2.

cosθ = b/a = 35.35 / 50.0
cosθ = 0.707
θ = acos(0.707) = +/-45°



Solving gives two possible values for Theta (+/-45°). This ambiguity can be resolved using additional AOA measurements from other antennas. The simplest approach is to have a third antenna located off-axis from the other two, from which another AOA can be computed and the intersection performed.

Applications which perform this type of analysis are encouraged to consider performing their own analysis and outputting the resulting angles and distances, as demonstrated in example *10.10*

10.8.1. Timing analysis details

It should be emphasized that this specification does not dictate precisely when in the 802.11 packet reception the synchronized TDOA_CLOCK sensor should be set. This is an implementation detail left to the collection system. As such, any application that makes use of the TDOA_CLOCK field **must** also store an AppId that can be used to differentiate products. This will help avoid mistakenly comparing clock values which are not actually synchronized.

10.9. AOA example

The PPI-GEOLOCATION specification allows applications to encode synthesized values, such as the angle of arrival or the computed position of a transmitter. Both of these features are accomplished by using the appropriate vector characteristics.

In the following example, an AOA is computed from a single location (perhaps by utilizing a phased array system). Setting the appropriate vector characteristic in the vector tag is used to indicate than the vector represents an AOA.

Applications are free to synthesize AOA data from whatever source(s) they feel are appropriate. As such, applications that set the AOA vector characteristic **must** also include an AppId, to help differentiate results.

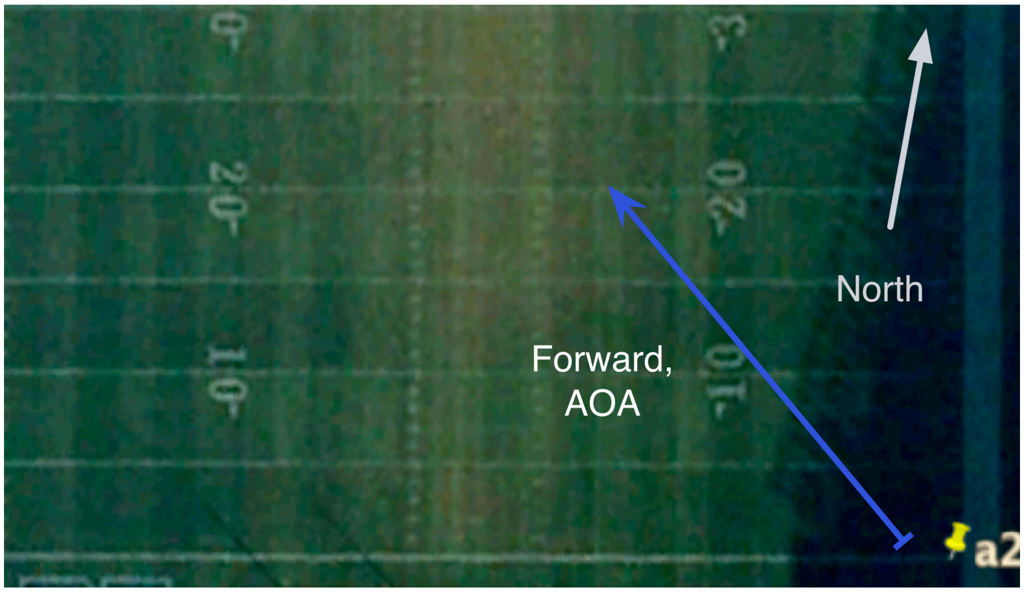
GPS-TAG (ant2)	
GPS-Flags	0x10 7:Manual input mode
Latitude	41.861904
Longitude	-87.616350
GPSTime	Tue May 17 09:40:32
FractionalTime	.20

VECTOR (antenna orientation)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x09 0: Antenna
Rot-X: (Pitch)	90.0
DescriptionString	Antenna-1 orientation

VECTOR (**Angle of arrival**)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x08 3: **Angle of arrival**
Rot-Z: (Heading)	323.4
Err-rot:	10.0 (10 degrees margin of error)
AppId	0x04030201

ANTENNA-TAG 2	
Gain	5 dBi
HorizBW	360.0
AntennaFlags	Bit 1: Horizontal-Polarity 1
Modelname	8dBi-MagMountOmni
DescriptionString	“Bottom Right of field”

802.11COMMON1 (ant1 signal)	
Antsignal	-80 dB



Providing Angles of arrival and signal strength can be useful to applications which not only need to know the direction of a transmitter, but how strong the signal is from a given position.

10.10. TRANSMITTER_POSITION example

Similar to AOA, applications can choose to encode not just the position of the collection system and the angle to the transmitter, but they can choose to encode what they have computed to be the position of the transmitter. In the following example the collection system has computed a position of a transmitter, and is encoding the distance to it from the collection system as an offset.40

GPS-TAG (ant2)	
GPS-Flags	0x10 7:Manual input mode
Latitude	41.861904
Longitude	-87.616350
GPSTime	Tue May 17 09:40:32
FractionalTime	.20
AppId	0x04030201

VECTOR (antenna orientation)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x1001 0: Antenna 12: Human derived
Rot-X: (Pitch)	90.0
DescriptionString	Antenna-1 orientation
AppId	0x04030201

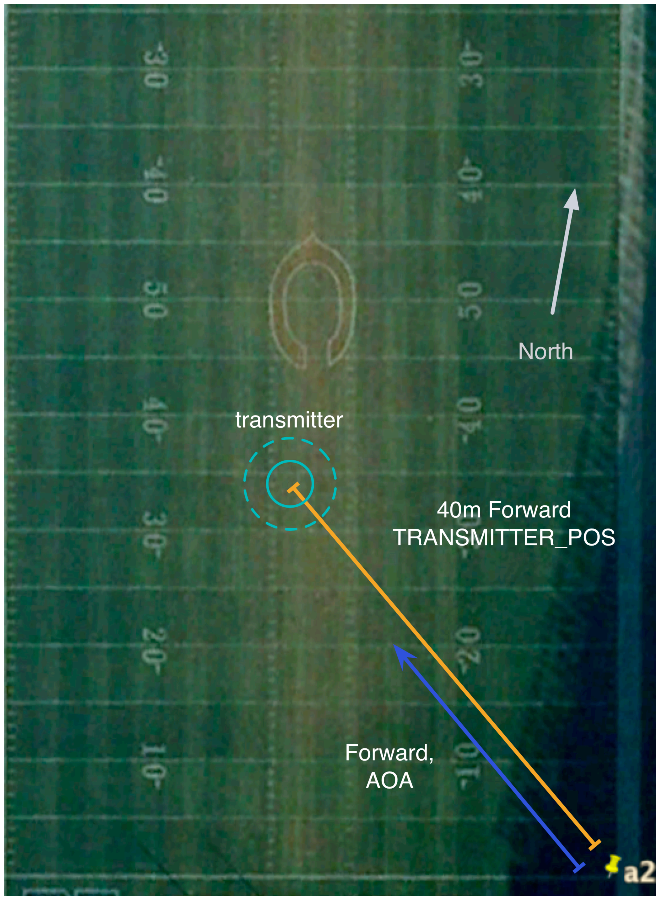
VECTOR (**Forward, Angle of arrival**)	
VectorFlags	0x02 RelativeTo:Earth (Rotations in E/N/U Coordinates)
VectorChars	0x09 1: **Defines_Forward** 3: **Angle of arrival**
Rot-Z: (Heading)	323.4
Err-Rot:	10.0
AppId	0x04030201

VECTOR (**Transmitter Position**)	
VectorFlags	0x00 RelativeTo:**Forward**
VectorChars	0x10 5:TRANSMITTER_POSITION
Off-Y: (Forward)	40 meters
Err-Off:	2.0 meters
AppId	0x04030201

ANTENNA-TAG 2	
Gain	5 dBi
HorizBW	360.0
AntennaFlags	Bit 1: Horizontal-Polarity 1

Modelname	8dBi-MagMountOmni
zDescriptionString	"Bottom Right of field"
AppId	0x04030201

802.11COMMON1 (ant1 signal)	
Antsignal	-80 dB



This set of tags contains the most useful set of information. It allows the GEOLOCATION-ENGINE to know where the packet was captured, what angle it was captured at, the computed distance to the transmitter, **and** the signal strength the packet was received at when it arrived at t2. This data also contains error estimates on the angle of arrive (2nd vetor tag, err_rot), and distance to the transmitter (3rd vector err_off).

Applications that perform TDOA analysis are strongly encouraged to encode their results in this format.

10.11. Dead reckoning with accelerometers

PPI-GEOLOCATION tags can support accelerometer-based applications in two manners. In the first case, the application itself performs integration on the acceleration data, and outputs packets tagged at the appropriate location. This can be accomplished either with GPS-tags, or with Vector offsets. This case is similar enough to those previously presented to need no further explanation.

Alternately, the application can output a stream of sensor data that is used by a GEOLOCATION-TAG processing engine to compute the position a packet was captured at. While it is possible to encode all of the relevant information into a PPI-GEOLOCATION tagged packet capture, doing so requires the processing application to remember state across individual packets. As such, applications that wish to perform this integration go above and beyond the rules specified in the PPI-GEOLOCATION state machine, and therefore may be application specific.

11. Implementers notes

The following section is intended to help implementers answer common questions about the GEOLOCATION-TAG specification.

11.1. What is the current version of this specification?

2.0.0 represents the first widespread distribution of this specification. New applications should use this version, and should set the geotag_ver field to 2 as well.

11.2. My GPS outputs a heading and speed. Why can't it go in the GPS TAG?

All heading information should be stored in a VECTOR-TAG. Please craft a VECTOR-TAG, fill in the Rot-Z (Heading) field, and set the VectorCharacteristics.`DIRECTION_OF _TRAVEL` bit appropriately. You should also set Bit 8 in VectorCharacteristics (GPS_ DERIVED).
Velocity is stored in Sensor tags. Please see 10.3 for details on this use case.

11.3. So I've got this VECTOR-TAG. What does it represent?

Vectors can represent many things (vehicle orientation, antenna orientation, etc).
Check the VectorCharacteristics field to be sure. Alternately, just utilize the provided state machine and it will handle this all for you by providing a frame of reference for each characteristic.

11.4. When should I set the DEFINES_FORWARD bit?

Any application that wishes to define its own local coordinate system must set the DEFINES_FORWARD bit to initialize it. Example 10.4 illustrates the most intuitive use of the forward frame of reference. Example 10.6 illustrates another use of the forward frame of reference.

11.5. Can a single packet set DEFINES_FORWARD more than once?

Yes. If an application decides that it easier to encode a system with more than one R/F/U coordinate system it may redefine Forward_Frame at any time. Applications that find themselves doing this may find it more convenient to encode information RelativeTo:CurrentFrame.

11.6. I've got this super-fancy TDOA based system. How is it supported by this specification?

TDOA based systems can either encode the relevant position/timing information and allow a post-processor to perform the triangulation work. (See Ex 10.8), or they can encode the results of the triangulation. *If possible, applications that perform this sort of geo-location are encouraged to perform their own triangulation and to encode their results in a manner similar to that presented in 10.10*

11.7. I've got this super-fancy accelerometer based system. Can I encode it with this format?

Applications that perform dead-reckoning with accelerometers that wish to support this specification are encouraged to perform integration at the application level and encode the results in a GPS-TAG or vector offset.

Applications **may** encode raw acceleration information as well as heading information using Sensor tags, however the specification does not specify a standard way to handle the state that much be stored across individual packets. Applications that pursue this direction are encouraged to contact the maintainer of this specification, so that a cross-platform solution can be agreed upon.

11.8. *I want to encode some data in PPI-GEOLOCATION TAG, but don't have a packet to insert it into. What should I do?*

Many applications need to write a steady stream of sensor data (position, acceleration, time, etc). These applications are encouraged to produce PPI-GEOLOCATION tagged packets with 0 in the pfh_datalen field.

12. Wireshark dissector examples

Wireshark contains dissectors for all of the described PPI-GEOLOCATION tags. This makes it a valuable tool in development and testing of PPI-GEOLOCATION aware applications. However, it should be emphasized that the included dissctors do not implement a PPI-GEOLOCATION state machine. Therefore there is no easy way to perform vector operations reliably, or to query the state of a PPI-GEOLOCATION tag state machine as is parses the input packet. The following examples are for illustration only; **proper interpretation of a PPI-GEOLOCATION tagged packet requires the implementation of a PPI-GEOLOCATION state machine**.

12.1. *Basic geo-fencing*

Basic geo-fencing can be performed using a display filter with wireshark. It should be emphasized that wireshark does not contain a compliant ppi-geolocation state machine, and therefore there is no simple way to perform the appropriate vector operations (for example, offsets). However, rough filtering based on GPS tags can be performed in the following manner.

```
((ppi_gps.lon <= -155.01) && (ppi_gps.lon >= -155.03) &&
 (ppi_gps.lat >=   19.01) && (ppi_gps.lat <=   19.03))
```

12.2. *Beamwidth filtering*

Another useful filter: Only looking at data captured with an Omni-directional antenna where the packet was received with a signal value > -75 db.

```
(ppi_antenna.horizbw == 360) && (radiotap.dbm_antsignal > -75)
```

Or, if dot11common tags are in use

```
(ppi_antenna.horizbw == 360) && (ppi.80211-common.dbm.antsignal > -75)
```

12.3. *Heading filtering*

The following filter illustrates the functionally of a VECTOR TAG by only displaying packets that were captured on an antenna which was pointed Northeasterly.

```
(ppi_vector.vflags.relative_to == 0x01)&& ppi_vector.chars.antenna == 1)
(ppi_vector.err_rot <= 50) && (ppi_antenna.horizbw < 360) &&
(ppi_vector.heading <= 60 && ppi_vector.heading >= 30) &&
```

The first line line explicitly checks for rotational coordinates that are Relativeto: Earth, and that the vector represents an antenna. The test for a reasonably low rotation error is to avoid processing packets that may have been tagged with to much error (perhaps magnetic interference with a digital compass). The beamwidth test is to make sure we are only processing packets with a directional (not Omni-directional) antenna.

This example shows the limitations of processing PPI-GEOLOCATION tags without an appropriate state machine. It is difficult to write a filter that can perform a similar check if the antenna Vector is RelativeTo:Forward or Current.

13. Future work

While this specification hopes to provide a solid base for interoperability among vendors, it has some known limitations. The following issues will be addressed in the future. *Vendors who are interested in any of these areas are encouraged to contact the maintainer of this specification.*

13.1. MIMO support.

Although the current specification explicitly handles multiple antenna and signal strength fields, the current support is based around duplicate packets being captured on independent interfaces. MIMO support could be added by indicating that each Antenna, 802.11COMMON tag par indicates a spatial stream. The other option is an attempt at standardizing the CACE 802.11N related tags interaction with the GEOLOCATION-STATE machine. Both options are currently under consideration.

Vendors with concerns or suggestions regarding the best way to integrate MIMO support are encourage to contact the maintainer of this specification.

13.2. Accelerometer based support

Acceleration information can be stored in Sensor-tags. However, the current iteration of the specification does not dictate how state should be carried across packets. Currently, the best way for accelerometer based applications to utilize this specification is to perform the integration at the application level and output data that is already tagged with the computed position.

.

14. Administrivia

Inquiries, comments, and any other feedback regarding this standard should be directed to jellch@harris.com

14.1. Future updates

The current maintainers reserve the right to make minor changes to the GEOLOCATION-TAG processing algorithms as implementations proceed. These changes will **not** cause a version bump in the base geolocation-tag header. **Only updates that would cause parsing errors, or the addition of very significant fields will increment the version field.**

14.2. Authors

This specification was primarily authored by Jon Ellch of Harris corporation. It was technically reviewed by Mike Troutman and Bob Riemenschnieder.

14.3. Revision history

2.0.0: Added sensor tag, cleaned up state machine, moved velocity/acceleration.
1.2.0: Dropped all minor revisions. This should simply be called 1.2.0
1.2-d20: Draft widely circulated;

14.4. References

1) CACE PPI specification, http://www.cacetech.com/documents/
2) Radiotap specification, http://www.radiotap.org/