 A Means for Expressing Location Information in the Domain Name System

Status of this Memo

1. Abstract

   This memo defines a new DNS RR type for experimental purposes. This
   RFC describes a mechanism to allow the DNS to carry location
   information about hosts, networks, and subnets. Such information for
   a small subset of hosts is currently contained in the flat-file UUCP
   maps. However, just as the DNS replaced the use of HOSTS.TXT to
   carry host and network address information, it is possible to replace
   the UUCP maps as carriers of location information.

   This RFC defines the format of a new Resource Record (RR) for the
   Domain Name System (DNS), and reserves a corresponding DNS type
   mnemonic (LOC) and numerical code (29).

   This RFC assumes that the reader is familiar with the DNS [RFC 1034,
   RFC 1035]. The data shown in our examples is for pedagogical use and
   does not necessarily reflect the real Internet.

2. RDATA Format

```
       MSB                                           LSB
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      0|        VERSION        |         SIZE          |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      2|       HORIZ PRE       |       VERT PRE        |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      4|                   LATITUDE                    |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      6|                   LATITUDE                    |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      8|                   LONGITUDE                   |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     10|                   LONGITUDE                   |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     12|                   ALTITUDE                    |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
     14|                   ALTITUDE                    |
       +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    (octet)
```
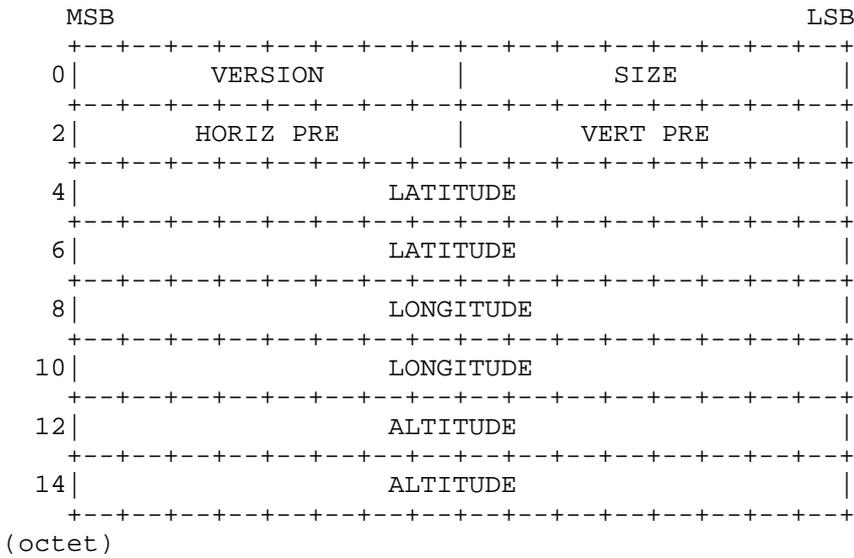
where:

VERSION   Version number of the representation. This must be zero.
             Implementations are required to check this field and make
             no assumptions about the format of unrecognized versions.

SIZE   The diameter of a sphere enclosing the described entity, in
             centimeters, expressed as a pair of four-bit unsigned
             integers, each ranging from zero to nine, with the most
             significant four bits representing the base and the second
             number representing the power of ten by which to multiply
             the base. This allows sizes from 0e0 (<1cm) to 9e9
             (90,000km) to be expressed. This representation was chosen
             such that the hexadecimal representation can be read by
             eye; 0x15 = 1e5. Four-bit values greater than 9 are
             undefined, as are values with a base of zero and a non-zero
             exponent.

             Since 20000000m (represented by the value 0x29) is greater
             than the equatorial diameter of the WGS 84 ellipsoid
             (12756274m), it is therefore suitable for use as a
             "worldwide" size.

HORIZ PRE   The horizontal precision of the data, in centimeters,
             expressed using the same representation as SIZE. This is
             the diameter of the horizontal "circle of error", rather

than a "plus or minus" value. (This was chosen to match
the interpretation of SIZE; to get a "plus or minus" value,
divide by 2.)

VERT PRE   The vertical precision of the data, in centimeters,
expressed using the sane representation as for SIZE. This
is the total potential vertical error, rather than a "plus
or minus" value. (This was chosen to match the
interpretation of SIZE; to get a "plus or minus" value,
divide by 2.) Note that if altitude above or below sea
level is used as an approximation for altitude relative to
the [WGS 84] ellipsoid, the precision value should be
adjusted.

LATITUDE   The latitude of the center of the sphere described by the
SIZE field, expressed as a 32-bit integer, most significant
octet first (network standard byte order), in thousandths
of a second of arc. 2^31 represents the equator; numbers
above that are north latitude.

LONGITUDE   The longitude of the center of the sphere described by the
SIZE field, expressed as a 32-bit integer, most significant
octet first (network standard byte order), in thousandths
of a second of arc, rounded away from the prime meridian.
2^31 represents the prime meridian; numbers above that are
east longitude.

ALTITUDE   The altitude of the center of the sphere described by the
SIZE field, expressed as a 32-bit integer, most significant
octet first (network standard byte order), in centimeters,
from a base of 100,000m below the [WGS 84] reference
spheroid used by GPS (semimajor axis a=6378137.0,
reciprocal flattening rf=298.257223563). Altitude above
(or below) sea level may be used as an approximation of
altitude relative to the the [WGS 84] spheroid, though due
to the Earth's surface not being a perfect spheroid, there
will be differences. (For example, the geoid (which sea
level approximates) for the continental US ranges from 10
meters to 50 meters below the [WGS 84] spheroid.
Adjustments to ALTITUDE and/or VERT PRE will be necessary
in most cases. The Defense Mapping Agency publishes geoid
height values relative to the [WGS 84] ellipsoid.

3. Master File Format

   The LOC record is expressed in a master file in the following format:

   <owner> <TTL> <class> LOC (d1 [m1 [s1]] {"N"|"S"} d2 [m2 [s2]]
                                 {"E"|"W"} alt["m"] [siz["m"] [hp["m"]
                                 [vp["m"]]]])

   (The parentheses are used for multi-line data as specified in [RFC
   1035] section 5.1.)

   where:

       d1:   [0 .. 90]   (degrees latitude)
       d2:   [0 .. 180]   (degrees longitude)
       m1, m2: [0 .. 59]   (minutes latitude/longitude)
       s1, s2: [0 .. 59.999]   (seconds latitude/longitude)
       alt: [-100000.00 .. 42849672.95] BY .01 (altitude in meters)
       siz, hp, vp: [0 .. 90000000.00] (size/precision in meters)

   If omitted, minutes and seconds default to zero, size defaults to 1m,
   horizontal precision defaults to 10000m, and vertical precision
   defaults to 10m. These defaults are chosen to represent typical
   ZIP/postal code area sizes, since it is often easy to find
   approximate geographical location by ZIP/postal code.

4. Example Data

;;;
;;; note that these data would not all appear in one zone file
;;;

;; network LOC RR derived from ZIP data. note use of precision defaults
cambridge-net.kei.com.   LOC   42 21 54 N 71 06 18 W -24m 30m

;; higher-precision host LOC RR. note use of vertical precision default
loiosh.kei.com.   LOC   42 21 43.952 N 71 5 6.344 W
                                     -24m 1m 200m

pipex.net.   LOC   52 14 05 N 00 08 50 E 10m

curtin.edu.au.   LOC   32 7 19 S 116 2 25 E 10m

rwy04L.logan-airport.boston.   LOC   42 21 28.764 N 71 00 51.617 W
                                     -44m 2000m

5. Application use of the LOC RR

5.1 Suggested Uses

   Some uses for the LOC RR have already been suggested, including the
   USENET backbone flow maps, a "visual traceroute" application showing
   the geographical path of an IP packet, and network management
   applications that could use LOC RRs to generate a map of hosts and
   routers being managed.

5.2 Search Algorithms

   This section specifies how to use the DNS to translate domain names
   and/or IP addresses into location information.

   If an application wishes to have a "fallback" behavior, displaying a
   less precise or larger area when a host does not have an associated
   LOC RR, it MAY support use of the algorithm in section 5.2.3, as
   noted in sections 5.2.1 and 5.2.2. If fallback is desired, this
   behaviour is the RECOMMENDED default, but in some cases it may need
   to be modified based on the specific requirements of the application
   involved.

   This search algorithm is designed to allow network administrators to
   specify the location of a network or subnet without requiring LOC RR
   data for each individual host. For example, a computer lab with 24
   workstations, all of which are on the same subnet and in basically
   the same location, would only need a LOC RR for the subnet.
   (However, if the file server's location has been more precisely
   measured, a separate LOC RR for it can be placed in the DNS.)

5.2.1 Searching by Name

   If the application is beginning with a name, rather than an IP
   address (as the USENET backbone flow maps do), it MUST check for a
   LOC RR associated with that name. (CNAME records should be followed
   as for any other RR type.)

   If there is no LOC RR for that name, all A records (if any)
   associated with the name MAY be checked for network (or subnet) LOC
   RRs using the "Searching by Network or Subnet" algorithm (5.2.3). If
   multiple A records exist and have associated network or subnet LOC
   RRs, the application may choose to use any, some, or all of the LOC
   RRs found, possibly in combination. It is suggested that multi-homed
   hosts have LOC RRs for their name in the DNS to avoid any ambiguity
   in these cases.

Note that domain names that do not have associated A records must
have a LOC RR associated with their name in order for location
information to be accessible.

5.2.2 Searching by Address

If the application is beginning with an IP address (as a "visual
traceroute" application might be) it MUST first map the address to a
name using the IN-ADDR.ARPA namespace (see [RFC 1034], section
5.2.1), then check for a LOC RR associated with that name.

If there is no LOC RR for the name, the address MAY be checked for
network (or subnet) LOC RRs using the "Searching by Network or
Subnet" algorithm (5.2.3).

5.2.3 Searching by Network or Subnet

Even if a host's name does not have any associated LOC RRs, the
network(s) or subnet(s) it is on may. If the application wishes to
search for such less specific data, the following algorithm SHOULD be
followed to find a network or subnet LOC RR associated with the IP
address. This algorithm is adapted slightly from that specified in
[RFC 1101], sections 4.3 and 4.4.

Since subnet LOC RRs are (if present) more specific than network LOC
RRs, it is best to use them if available. In order to do so, we
build a stack of network and subnet names found while performing the
[RFC 1101] search, then work our way down the stack until a LOC RR is
found.

1. create a host-zero address using the network portion of the IP
   address (one, two, or three bytes for class A, B, or C networks,
   respectively). For example, for the host 128.9.2.17, on the class
   B network 128.9, this would result in the address "128.9.0.0".

2. Reverse the octets, suffix IN-ADDR.ARPA, and query for PTR and A
   records. Retrieve:

               0.0.9.128.IN-ADDR.ARPA. PTR   isi-net.isi.edu.
                                        A   255.255.255.0

   Push the name "isi-net.isi.edu" onto the stack of names to be
   searched for LOC RRs later.

   3. Since an A RR was found, repeat using mask from RR
      (255.255.255.0), constructing a query for 0.2.9.128.IN-ADDR.ARPA.
      Retrieve:

              0.2.9.128.IN-ADDR.ARPA. PTR   div2-subnet.isi.edu.
                                       A   255.255.255.240

      Push the name "div2-subnet.isi.edu" onto the stack of names to be
      searched for LOC RRs later.

   4. Since another A RR was found, repeat using mask 255.255.255.240
      (x'FFFFFFF0'), constructing a query for 16.2.9.128.IN-ADDR.ARPA.
      Retrieve:

              16.2.9.128.IN-ADDR.ARPA. PTR   inc-subsubnet.isi.edu.

      Push the name "inc-subsubnet.isi.edu" onto the stack of names to
      be searched for LOC RRs later.

   5. Since no A RR is present at 16.2.9.128.IN-ADDR.ARPA., there are no
      more subnet levels to search. We now pop the top name from the
      stack and check for an associated LOC RR. Repeat until a LOC RR
      is found.

      In this case, assume that inc-subsubnet.isi.edu does not have an
      associated LOC RR, but that div2-subnet.isi.edu does. We will
      then use div2-subnet.isi.edu's LOC RR as an approximation of this
      host's location. (Note that even if isi-net.isi.edu has a LOC RR,
      it will not be used if a subnet also has a LOC RR.)

5.3 Applicability to non-IN Classes and non-IP Addresses

   The LOC record is defined for all RR classes, and may be used with
   non-IN classes such as HS and CH. The semantics of such use are not
   defined by this memo.

   The search algorithm in section 5.2.3 may be adapted to other
   addressing schemes by extending [RFC 1101]'s encoding of network
   names to cover those schemes. Such extensions are not defined by
   this memo.

6. References

   [RFC 1034] Mockapetris, P., "Domain Names - Concepts and Facilities",
              STD 13, RFC 1034, USC/Information Sciences Institute,
              November 1987.

   [RFC 1035] Mockapetris, P., "Domain Names - Implementation and
              Specification", STD 13, RFC 1035, USC/Information Sciences
              Institute, November 1987.

   [RFC 1101] Mockapetris, P., "DNS Encoding of Network Names and Other
              Types", RFC 1101, USC/Information Sciences Institute,
              April 1989.

   [WGS 84] United States Department of Defense; DoD WGS-1984 - Its
            Definition and Relationships with Local Geodetic Systems;
            Washington, D.C.; 1985; Report AD-A188 815 DMA; 6127; 7-R-
            138-R; CV, KV;

7. Security Considerations

   High-precision LOC RR information could be used to plan a penetration
   of physical security, leading to potential denial-of-machine attacks.
   To avoid any appearance of suggesting this method to potential
   attackers, we declined the opportunity to name this RR "ICBM".

8. Authors' Addresses

   The authors as a group can be reached as <loc@pipex.net>.

   Christopher Davis
   Kapor Enterprises, Inc.
   238 Main Street, Suite 400
   Cambridge, MA 02142

   Phone: +1 617 576 4532
   EMail: ckd@kei.com

   Paul Vixie
   Vixie Enterprises
   Star Route Box 159A
   Woodside, CA 94062

   Phone: +1 415 747 0204
   EMail: paul@vix.com

    Tim Goodwin
    Public IP Exchange Ltd (PIPEX)
    216 The Science Park
    Cambridge CB4 4WA
    UK

    Phone: +44 1223 250250
    EMail: tim@pipex.net

    Ian Dickinson
    FORE Systems
    2475 The Crescent
    Solihull Parkway
    Birmingham Business Park
    B37 7YE
    UK

    Phone: +44 121 717 4444
    EMail: idickins@fore.co.uk

Appendix A: Sample Conversion Routines

```
/*
 * routines to convert between on-the-wire RR format and zone file
 * format.  Does not contain conversion to/from decimal degrees;
 * divide or multiply by 60*60*1000 for that.
 */

static unsigned int poweroften[10] = {1, 10, 100, 1000, 10000, 100000,
                            1000000,10000000,100000000,1000000000};

/* takes an XeY precision/size value, returns a string representation.*/
static const char *
precsize_ntoa(prec)
        u_int8_t prec;
{
        static char retbuf[sizeof("90000000.00")];
        unsigned long val;
        int mantissa, exponent;

        mantissa = (int)((prec >> 4) & 0x0f) % 10;
        exponent = (int)((prec >> 0) & 0x0f) % 10;

        val = mantissa * poweroften[exponent];

        (void) sprintf(retbuf,"%d.%.2d", val/100, val%100);
        return (retbuf);
}

/* converts ascii size/precision X * 10**Y(cm) to 0xXY. moves pointer.*/
static u_int8_t
precsize_aton(strptr)
        char **strptr;
{
        unsigned int mval = 0, cmval = 0;
        u_int8_t retval = 0;
        register char *cp;
        register int exponent;
        register int mantissa;

        cp = *strptr;

        while (isdigit(*cp))
                mval = mval * 10 + (*cp++ - '0');

        if (*cp == '.') {               /* centimeters */
                cp++;
                if (isdigit(*cp)) {
```

```
                        cmval = (*cp++ - '0') * 10;
                        if (isdigit(*cp)) {
                                cmval += (*cp++ - '0');
                        }
                }
        }
        cmval = (mval * 100) + cmval;

        for (exponent = 0; exponent < 9; exponent++)
                if (cmval < poweroften[exponent+1])
                        break;

        mantissa = cmval / poweroften[exponent];
        if (mantissa > 9)
                mantissa = 9;

        retval = (mantissa << 4) | exponent;

        *strptr = cp;

        return (retval);
}

/* converts ascii lat/lon to unsigned encoded 32-bit number.
 *  moves pointer. */
static u_int32_t
latlon2ul(latlonstrptr,which)
        char **latlonstrptr;
        int *which;
{
        register char *cp;
        u_int32_t retval;
        int deg = 0, min = 0, secs = 0, secsfrac = 0;

        cp = *latlonstrptr;

        while (isdigit(*cp))
                deg = deg * 10 + (*cp++ - '0');

        while (isspace(*cp))
                cp++;

        if (!(isdigit(*cp)))
                goto fndhemi;

        while (isdigit(*cp))
                min = min * 10 + (*cp++ - '0');
```

```
        while (isspace(*cp))
                cp++;

        if (!(isdigit(*cp)))
                goto fndhemi;

        while (isdigit(*cp))
                secs = secs * 10 + (*cp++ - '0');

        if (*cp == '.') {                /* decimal seconds */
                cp++;
                if (isdigit(*cp)) {
                        secsfrac = (*cp++ - '0') * 100;
                        if (isdigit(*cp)) {
                                secsfrac += (*cp++ - '0') * 10;
                                if (isdigit(*cp)) {
                                        secsfrac += (*cp++ - '0');
                                }
                        }
                }
        }

        while (!isspace(*cp))   /* if any trailing garbage */
                cp++;

        while (isspace(*cp))
                cp++;

 fndhemi:
        switch (*cp) {
        case 'N': case 'n':
        case 'E': case 'e':
                retval = ((unsigned)1<<31)
                        + (((((deg * 60) + min) * 60) + secs) * 1000)
                        + secsfrac;
                break;
        case 'S': case 's':
        case 'W': case 'w':
                retval = ((unsigned)1<<31)
                        - (((((deg * 60) + min) * 60) + secs) * 1000)
                        - secsfrac;
                break;
        default:
                retval = 0;     /* invalid value -- indicates error */
                break;
        }

        switch (*cp) {
```

```
        case 'N': case 'n':
        case 'S': case 's':
                *which = 1;      /* latitude */
                break;
        case 'E': case 'e':
        case 'W': case 'w':
                *which = 2;      /* longitude */
                break;
        default:
                *which = 0;      /* error */
                break;
        }

        cp++;                            /* skip the hemisphere */

        while (!isspace(*cp))    /* if any trailing garbage */
                cp++;

        while (isspace(*cp))     /* move to next field */
                cp++;

        *latlonstrptr = cp;

        return (retval);
}

/* converts a zone file representation in a string to an RDATA
 * on-the-wire representation. */
u_int32_t
loc_aton(ascii, binary)
        const char *ascii;
        u_char *binary;
{
        const char *cp, *maxcp;
        u_char *bcp;

        u_int32_t latit = 0, longit = 0, alt = 0;
        u_int32_t lltemp1 = 0, lltemp2 = 0;
        int altmeters = 0, altfrac = 0, altsign = 1;
        u_int8_t hp = 0x16;     /* default = 1e6 cm = 10000.00m = 10km */
        u_int8_t vp = 0x13;     /* default = 1e3 cm = 10.00m */
        u_int8_t siz = 0x12;    /* default = 1e2 cm = 1.00m */
        int which1 = 0, which2 = 0;

        cp = ascii;
        maxcp = cp + strlen(ascii);

        lltemp1 = latlon2ul(&cp, &which1);
```

```
        lltemp2 = latlon2ul(&cp, &which2);

        switch (which1 + which2) {
        case 3:                 /* 1 + 2, the only valid combination */
            if ((which1 == 1) && (which2 == 2)) { /* normal case */
                    latit = lltemp1;
                    longit = lltemp2;
            } else if ((which1 == 2) && (which2 == 1)) {/*reversed*/
                    longit = lltemp1;
                    latit = lltemp2;
            } else {        /* some kind of brokenness */
                    return 0;
            }
            break;
        default:                /* we didn't get one of each */
            return 0;
        }

        /* altitude */
        if (*cp == '-') {
                altsign = -1;
                cp++;
        }

        if (*cp == '+')
                cp++;

        while (isdigit(*cp))
                altmeters = altmeters * 10 + (*cp++ - '0');

        if (*cp == '.') {               /* decimal meters */
                cp++;
                if (isdigit(*cp)) {
                        altfrac = (*cp++ - '0') * 10;
                        if (isdigit(*cp)) {
                                altfrac += (*cp++ - '0');
                        }
                }
        }

        alt = (10000000 + (altsign * (altmeters * 100 + altfrac)));

        while (!isspace(*cp) && (cp < maxcp))
                                            /* if trailing garbage or m */
                cp++;

        while (isspace(*cp) && (cp < maxcp))
                cp++;
```

```
        if (cp >= maxcp)
                goto defaults;

        siz = precsize_aton(&cp);

        while (!isspace(*cp) && (cp < maxcp))/*if trailing garbage or m*/
                cp++;

        while (isspace(*cp) && (cp < maxcp))
                cp++;

        if (cp >= maxcp)
                goto defaults;

        hp = precsize_aton(&cp);

        while (!isspace(*cp) && (cp < maxcp))/*if trailing garbage or m*/
                cp++;

        while (isspace(*cp) && (cp < maxcp))
                cp++;

        if (cp >= maxcp)
                goto defaults;

        vp = precsize_aton(&cp);

 defaults:

        bcp = binary;
        *bcp++ = (u_int8_t) 0;  /* version byte */
        *bcp++ = siz;
        *bcp++ = hp;
        *bcp++ = vp;
        PUTLONG(latit,bcp);
        PUTLONG(longit,bcp);
        PUTLONG(alt,bcp);

        return (16);            /* size of RR in octets */
}

/* takes an on-the-wire LOC RR and prints it in zone file
 * (human readable) format. */
char *
loc_ntoa(binary,ascii)
        const u_char *binary;
        char *ascii;
{
```

```
        static char tmpbuf[255*3];

        register char *cp;
        register const u_char *rcp;

        int latdeg, latmin, latsec, latsecfrac;
        int longdeg, longmin, longsec, longsecfrac;
        char northsouth, eastwest;
        int altmeters, altfrac, altsign;

        const int referencealt = 100000 * 100;

        int32_t latval, longval, altval;
        u_int32_t templ;
        u_int8_t sizeval, hpval, vpval, versionval;

        char *sizestr, *hpstr, *vpstr;

        rcp = binary;
        if (ascii)
                cp = ascii;
        else {
                cp = tmpbuf;
        }

        versionval = *rcp++;

        if (versionval) {
                sprintf(cp,"; error: unknown LOC RR version");
                return (cp);
        }

        sizeval = *rcp++;

        hpval = *rcp++;
        vpval = *rcp++;

        GETLONG(templ,rcp);
        latval = (templ - ((unsigned)1<<31));

        GETLONG(templ,rcp);
        longval = (templ - ((unsigned)1<<31));

        GETLONG(templ,rcp);
        if (templ < referencealt) { /* below WGS 84 spheroid */
                altval = referencealt - templ;
                altsign = -1;
        } else {
```

```
            altval = templ - referencealt;
            altsign = 1;
    }

    if (latval < 0) {
            northsouth = 'S';
            latval = -latval;
    }
    else
            northsouth = 'N';

    latsecfrac = latval % 1000;
    latval = latval / 1000;
    latsec = latval % 60;
    latval = latval / 60;
    latmin = latval % 60;
    latval = latval / 60;
    latdeg = latval;

    if (longval < 0) {
            eastwest = 'W';
            longval = -longval;
    }
    else
            eastwest = 'E';

    longsecfrac = longval % 1000;
    longval = longval / 1000;
    longsec = longval % 60;
    longval = longval / 60;
    longmin = longval % 60;
    longval = longval / 60;
    longdeg = longval;

    altfrac = altval % 100;
    altmeters = (altval / 100) * altsign;

    sizestr = savestr(precsize_ntoa(sizeval));
    hpstr = savestr(precsize_ntoa(hpval));
    vpstr = savestr(precsize_ntoa(vpval));

    sprintf(cp,
            "%d %.2d %.2d.%.3d %c %d %.2d %.2d.%.3d %c %d.%.2dm
            %sm %sm %sm",
            latdeg, latmin, latsec, latsecfrac, northsouth,
            longdeg, longmin, longsec, longsecfrac, eastwest,
            altmeters, altfrac, sizestr, hpstr, vpstr);
```

```
        free(sizestr);
        free(hpstr);
        free(vpstr);

        return (cp);
}
```