APRX Software Requirement Specification

Table of Contents

1 APRX Software Requirement Specification

This is *Requirement Specification* for a software serving in Amateur Radio APRS service.

Reader is assumed to be proficient with used terminology, and they are not usually explained here.

1.1 Purpose:

This describes algorithmic, IO-, and environmental requirements for a software doing any combination of following four tasks related to APRS service:

1. Listen on messages with a radio, and pass them to APRSIS network service
2. Listen on messages with a radio, and selectively re-send them on radio
3. Listen on messages with a radio, and selectively re-send them on radios on other frequencies
4. Receive messages from APRSIS network, and after selective filtering, send some of them on radio

Existing *aprx* software implements Receive-Only (Rx) IGate functionality, and the purpose of this paper is to map new things that it will need for extending functionality further.

1.2 Usage Environments:

The *aprx* software can be used in several kinds of environments to handle multiple tasks associated with local APRS network infrastructure tasks.

On following one should remember that amateur radio **transmitters** need a specially licensed owner/operator or a license themselves, but receivers do not need such:

1. License-free Receive-Only (RX) IGate, to add more “ears” to hear packets, and to pipe them to APRSIS. (Owner/operator has a license, but a receiver does not need special *transmitter license*.)
2. Licensed bidirectional IGate, selectively passing messages from radio channels to APRSIS, and from APRSIS to radio channels, but not repeating packets heard on a radio channel back to a radio channel.
3. Licensed bidirectional IGate plus selectively re-sending of packets heard on radio channels back to radio channels
4. Licensed system for selectively re-sending of packets heard on radio channels back to other radio channels, and this without bidirectional IGate service.
5. Licensed system for selectively re-sending of packets heard on radio channels back to radio channels, and doing with with “receive only” IGate, so passing information heard on radio channel to APRSIS, and not the other way at all.

In more common case, there is single radio and single TNC attached to digipeating (re-sending), in more challenging cases there are multiple receivers all around, and very few transmitters. Truly challenging systems operate on multiple radio channels. As single-TNC and single-radio systems are just simple special cases of these complex systems, and for the purpose of this software requirements we consider the complex ones:

1. 3 different frequencies in use, traffic is being relayed in between them, and the APRSIS network.
2. On each frequency there are multiple receivers, and one well placed transmitter.
3. Relaying from one frequency to other frequency may end up having different rules, than when re-sending on same frequency: Incoming packet retains traced paths, and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.

2 Treatment rules:

Generally: All receivers report what they hear straight to APRSIS, after small amount of filtering of junk messages, and things which explicitly state that they should not be sent to APRSIS.

2.1 Basic IGate rules:

General rules for these receiving filters are described here:

http://www.aprs-is.net/IGateDetails.aspx

Gate all packets heard on RF to the Internet (Rx-IGate) EXCEPT

1. 3rd party packets (data type '}') should have all before and including the data type stripped and then the packet should be processed again starting with step 1 again. There are cases like D-STAR gateway to APRS of D-STAR associated operator (radio) positions.
2. generic queries (data type '?').
3. packets with TCPIP, TCPXX, NOGATE, or RFONLY in the header, especially in those opened up from a 3rd party packets.

Gate message packets and associated posits to RF (Tx-IGate) if

1. the receiving station has been heard within range within a predefined time period (range defined as digi hops, distance, or both).
2. the sending station has not been heard via RF within a predefined time period (packets gated from the Internet by other stations are excluded from this test).
3. the sending station does not have TCPXX, NOGATE, or RFONLY in the header.
4. the receiving station has not been heard via the Internet within a predefined time period.

A station is said to be heard via the Internet if packets from the station contain TCPIP* or TCPXX* in the header or if gated (3rd party) packets are seen on RF gated by the station and containing TCPIP or TCPXX in the 3rd party header (in other words, the station is seen on RF as being an IGate).

Gate all packets to RF based on criteria set by the sysop (such as call-sign, object name, etc.).

Rx-IGate to APRSIS can use duplicate detection, and refuse to repeat same packet over and over again to APRSIS network.

With more advanced looking inside frames to be relayed, both the digipeater and Tx-IGate can use filtering rules, like “packet reports a position that is within my service area.”

From multiple receivers + single (or fewer) transmitter(s) follows, than when a more usual system does not hear what it sent out itself, this one will hear, and its receivers must have a way to ignore a frame it sent out itself a moment ago.

Without explicit “ignore what I just sent” filtering, an APRS packet will get reported twice to APRSIS:

rx ⇒ igate-to-aprsis + digi ⇒ tx ⇒ rx ⇒ igate-to-aprsis + digi (dupe filter stops)

Digipeating will use common packet duplication testing to sent similar frame out only once per given time interval (normally 30 seconds.)

An RF/Analog way to handle the “master-TX spoke this one, I will ignore it” could be use of audio subtones (American Motorola lingo: PL tone, otherwise known as CTCSS.) Digipeater transmitters have unique CTCSS subtone at each, and all receivers have subtone decoders. When they detect same subtone as their master has, they mute the receiver to data demodulator audio signal.

A third way would be to recognize their master transmitter callsign in AX.25 VIA path, or at FROM field, which presumes that the master transmitters will do TRACE mode adding of themselves on digipeated paths.

2.2 Low-Level Transmission Rules:

These rules control repeated transmissions of data that was sent a moment ago, and other basic transmitter control issues, like persistence. In particular the persistence is fine example of how to efficiently use radio channel, by sending multiple small frames in quick succession with same preamble and then be silent for longer time.

1. Duplication detector per transmitter: Digipeater and Tx-IGate will ignore packets finding a hit in this subsystem.
2. A candidate packet is then subjected to a number of filters, and if approved for it, the packet will be put on duplicate packet detection database (one for each transmitter.) See Digipeater Rules, below.
3. Because the system will hear the packets it sends out itself, there must be a global expiring storage for recently sent packets, which the receivers can then compare against. (Around 100 packets of 80-120 bytes each.) This storage gets a full copy of packet being sent out – a full AX.25 frame.

Also, transmitters should be kept in limited leash: Transmission queue is less than T seconds (< 5 ?), which needs some smart scheduling coding, when link from computer to TNC is considerably faster.

Original KISS interface is defined as “best effort”: if TNC is busy while host sends a frame, the frame may be discarded, and “upper layers” will resend. In APRS Digipeating, the upper layer sends the packet once, and then declares circa 30 second moratorium on packets with same payload.

2.3 Low-Level Receiving Rules:

1. Received AX.25 packet is compared against “my freshly sent packets” storage, and matched ones are dropped. (Case of one/few transmitters, and multiple receivers hearing them.)
2. Received packet is validated against AX.25 basic structure, invalid ones are dropped.
3. Received packet is validated against Rx-IGate rules, forbidden ones are dropped (like when a VIA-field contains invalid data.)
4. Packet may be rejected for Rx-IGate, but it may still be valid for digipeating! For example a 3^{rd} party frame is OK to digipeat, but not to Rx-IGate to APRSIS! Also some D-STAR to APRS gateways output 3^{rd} party frames, while the original frame is quite close to an APRS frame.

Divide packet rejection filters to common, and destination specific ones.

2.4 Additional Tx-IGate rules:

The Tx-IGate can have additional rules for control:

1. Multiple filters look inside the message, and can enforce a rule of “repeat only packets within my service area,” or to “limit passing message responses only to destinations within my service area”. Filter input syntax per javAPRSSrvr's adjunct filters with negation extension.
2. Basic gate filtering rules:
    1. the receiving station has been heard within range within a predefined time period (range defined as digi hops, distance, or both).
    2. the sending station has not been heard via RF within a predefined time period (packets gated from the Internet by other stations are excluded from this test).
    3. the sending station does not have TCPXX, NOGATE, or RFONLY in the header.
    4. the receiving station has not been heard via the Internet within a predefined time period.

    A station is said to be heard via the Internet if packets from the station contain TCPIP* or TCPXX* in the header or if gated (3rd-party) packets are seen on RF gated by the station and containing TCPIP or TCPXX in the 3rd-party header (in other words, the station is seen on RF as being an IGate).
3. Optionally wait a few seconds (like a random number of seconds in range of 1 to 5 seconds) before letting received packet out. This permits other systems to be faster than the Tx-IGate system, and thus to get their voice.

2.5 Digipeater Rules:

Digipeater will do following for each transmitter:

1. Compare candidate packet against duplicate filter, if found, then drop it. (Low-level transmission rules, number 1)
2. Count number of hops the message has so far done, and...
3. Figure out the number of hops the message has been requested to do (e.g. "OH2XYZ-1>APRS,OH2RDU*,WIDE7-5: ..." will report that there was request of 7 hops, so far 2 have been executed – one is shown on trace path.)
4. If either of previous ones are over any of configured limits, the packet is dropped.
5. FIXME: WIDEn-N/TRACEn-N treatment rules: By default treat both as TRACE, have an option to disable "WIDE-is-TRACE" mode. Possibly additional keywords for cross-band digipeating? (E.g. Bruninga said '6MTRSn-N' would be 'WIDEn-N' on 50 MHz APRS, and only there.)
6. Multiple filters look inside the message, and can enforce a rule of "repeat only packets within my service area."
7. FIXME: Cross frequency digipeating? Treat much like Tx-IGate? Relaying from one frequency to other frequency may end up having different rules, than when re-sending on same frequency: Incoming packet retains traced paths, and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.
8. Cross band relaying may need to add both an indication of "received on 2m", and transmitter identifier: "sent on 6m":
   "OH2XYZ-1>APRS,RX2M*,OH2RDK-6*,WIDE3-2: ..."

   This "source indication token" may not have anything to do with real receiver identifier, which is always shown on packets passed to APRSIS.

The MIC-e has a weird way to define same thing as normal packets do with
    SRCCALL-n>DEST,WIDE2-2: ...
The MIC-e way (on specification, practically nobody implements it) is:
    SRCCALL-n>DEST-2: ...

2.6 Duplicate Detector

Normal digipeater duplicate packet detection compares message source (with SSID), destination (without SSID!), and payload data against other packets in self-expiring storage called “duplicate detector”. Lifetime of this storage is commonly considered to be 30 seconds.

Practically the packet being compared at Duplicate Detector will be terminated at first CR or LF in the packet, and if there is a space character preceding the line end, also that is ignored when calculating duplication match. **However: The Space Characters are sent, if any are received, also when at the end of the packet!** (Some TNC:s have added one or two extra space characters on packets they digipeat...)

The “destination without SSID” rule comes from MIC-e specification, where a destination WIDE uses SSID to denote number of distribution hops. Hardly anybody implements it.

2.7 Radio Interface Statistics Telemetry

Current *aprx* software offers telemetry data on radio interfaces. It consists of following four things. Telemetry is reported to APRS-IS every 10 minutes:

1. Channel occupancy average in Erlangs over 1 minute interval, and presented as busiest 1 minute within the report interval. Where real measure of carrier presence on radio channel is not available, the value is derived from number of received AX.25 frame bytes plus a fixed Stetson-Harrison constant added per each packet for overheads. That is then divided by presumed channel modulation speed, and thus derived a figure somewhere in between 0.0 and 1.0.
2. Channel occupancy average in Erlangs over 10 minute interval. Same data source as above.
3. Count of received packets over 10 minutes.
4. Count of packets dropped for some reason during that 10 minute period.

Additional telemetry data points could be:

1. Number of transmitted packets over 10 minute interval
2. Number of packets IGate:d from APRSIS over 10 minute interval
3. Number of packets digipeated for this radio interface over 10 minute interval
4. Erlang calculations could include both Rx and Tx, but could also be separate.

2.8 Individual Call-Signs for Each Receiver, or Not?

Opinions are mixed on the question of having separate call-signs for each receiver (and transmitter), or not. Even the idea to use all 16 available SSIDs for a call-sign for something does get some opposition.

- There is no license fee in most countries for receivers, and there is no need to limit used call-signs only on those used for the site transmitters.
- There is apparently some format rule on APRSIS about what a “call-sign” can be, but it is rather lax: 6 alphanumerics + optional tail of: “-” (minus sign) and one or two alphanumerics. For example OH2XYZ-R1 style call-sign can have 36 different values before running out of variations on last character alone (A to Z, 0 to 9.)
- Transmitter call-signs are important, and there valid AX.25 format call-signs are mandatory.

On digipeater setup the receiver call-signs are invisible on RF. There only transmitter call-signs must be valid AX.25 addresses.

Transmitters should have positional beacons for them sent on correct position, and auxiliary elements like receivers could have their positions either real (when elsewhere), or actually placed near the primary Tx location so that they are separate on close enough zoomed map plot.

Using individual receiver identities (and associated net-beaconed positions near the real location) can give an idea of where the packet was heard, and possibly on which band. At least the *aprs.fi* is able to show the path along which the position was heard.

2.9 Beaconing

Smallest time interval available to position viewing at aprs.fi site is 15 minutes. A beacon interval longer than that will at times disappear from that view. Default view interval is 60 minutes.

Beacon transmission time **must not** be manually configured to fixed exact minute. There are large peaks in APRSIS traffic because of people are beaconing out every 5 minutes, and every 10 minutes, at exact 5/10 minutes. (Common happening with e.g. *digi_ned.)*

Beaconing system must be able to spread the requests over the entire cycle time (10 to 30 minutes) evenly. Even altering the total cycle time by up to 10% at random at the start of each cycle should be considered (and associated re-scheduling of all beacon events at every cycle start). All this to avoid multiple non-coordinated systems running at same rhythm. System that uses floating point mathematics to determine spherical distance in between two positions can simplify the distribution process by using float mathematics. Also all-integer algorithms exist (e.g. Bresenham's line plotting algorithm.)

```
float dt = (float)cycle_in_seconds;
for (int i = 0; i < number_of_beacons;++i) {
      beacon[i].tx_time = now + (i+1) * dt;
}
```

With only one beacon, it will go out at the end of the beacon cycle.

Receiver location beacons need only to be on APRSIS with additional TCPXX token, transmitter locations could be also on radio.

2.9.1 Radio Beaconing

“Tactical situation awareness” beaconing frequency could be 5-10 minutes, WB4APR does suggest at most 10 minutes interval. Actively moving systems will send positions more often. Transmit time spread algorithm must be used.

Minimum interval of beacon transmissions to radio should be 60 seconds. If more beacons need to be sent in this time period, use of Persistence parameter on TNCs (and KISS) should help: Send the beacons one after the other (up to 3?) during same transmitter activation, and without prolonged buffer times in between them. That is especially suitable for beacons *without* any sort of distribution lists.

Minimize the number of radio beacons!

2.9.2 Network beaconing

Network beaconing cycle time can be up to 30 minutes.

Network beaconing can also transmit positions and objects at much higher rate, than radio beaconing. Transmit time spread algorithm must be used.

Net-beacons could also be bursting similar to radio beacon Persistence – within a reason.

3 Configuration Language

System configuration language has several semi-conflicting requirements:

1. Easy to use
2. Minimal setup necessary for start
3. Sensible defaults
4. Self-documenting
5. Efficient self-diagnostics
6. Powerful – as ability to define complicated things

Examples of powerful, yet miserably complicated rule writing can be seen on *digi_ned*, and *aprsd*. Both have proven over and over again that a correct configuration is hard to make.

On Embedded front, things like UIDIGI have tens of parameters to set, many of which can be configured so that the network behaviour is degraded, if not downright broken.

UIView32 has poor documentation on what to put on destination address, and therefore many users put there “WIDE” instead of “APRS,WIDE”, and thus create broken beacons.

Current *aprx* configuration follows “minimal setup” and “easy to use” rules, it is even “self-documenting” and “self-diagnosing”, but its lack of power becomes apparent.

Some examples:

1. `radio serial   /dev/ttyUSB0  19200 8n1    KISS  callsign N0CALL-14`
2. ```
   netbeacon for N0CALL-13 dest "APRS" via “NOGATE” symbol "R&"
     lat "6016.30N" lon "02506.36E" comment "aprx - an Rx-only iGate"
   ```

The “radio serial” definition lacks handling of multiple TNCs using KISS device IDs, and there is no easy way to define subid/callsign pairs.

The “netbeacon” format can do only basic “!”-type location fix packets. Extending it to objects would probably cover 99% of wanted use cases.

Both have extremely long input lines, no input line folding is supported!
   ```

3.1 APRSIS Interface Definition

There can be multiple APRSIS connections defined, although only one is used at any time.

Parameter sets controlling this functionality is non-trivial.

```
<aprsis>                # Alternate A, single server, defaults
   login  OH2XYZ-R1
   server finland.aprs2.net:14580
   filter ....
   heartbeat-timeout 2 minutes
</aprsis>
<aprsis>                # Alternate B, multiple alternate servers
   login  OH2XYZ-R1
   <server finland.aprs2.net:14580>
      heartbeat-timeout 2 minutes
      filter ....
   </server>
   <server rotate.aprs.net:14580>
      heartbeat-timeout 120 seconds
      filter ....
      # Alt Login ?
   </server>
</aprsis>
```

3.2 Radio Interface Definitions

Interfaces are of multitude, some are just plain serial ports, some can be accessed via Linux internal AX.25 network, or by some other means, platform depending.

```
<interface>
   serial-device /dev/ttyUSB1 19200 8n1 KISS
   tx-ok         false       # receive only (default)
   callsign      OH2XYZ-R2   # KISS subif 0
</interface>
<interface>
   serial-device /dev/ttyUSB2 19200 8n1 KISS
   <kiss-subif 0>
      callsign OH2XYZ-2
      tx-ok    true          # This is our transmitter
   </kiss-subif>
   <kiss-subif 1>
      callsign OH2XYZ-R3     # This is receiver
      tx-ok    false         # receive only (default)
   </kiss-subif>
</interface>
<interface>
   ax25-device OH2XYZ-6
   tx-ok       true          # This is also transmitter
</interface>
```

3.3 Digipeating Definitions

The powerfulness is necessary for controlled digipeating, where traffic from multiple sources gets transmutated to multiple destinations, with different rules for each of them.

1. Destination device definition (refer to “serial radio” entry, or AX.25 network interface), must find a “tx-ok” feature flag on the interface definition.
2. Possible Tx-rate-limit parameters
3. Groups of:
    1. Source device references (of “serial radio” or ax25-rxport call-signs, or “APRSIS” keyword)
    2. Filter rules, if none are defined, source will not pass anything in. Can have also subtractive filters – “everything but not that”. Multiple filter entries are processed in sequence.
    3. Digipeat limits – max requests, max executed hops.
    4. Control of treat WIDEn-N as TRACEn-N or not. (Default: treat as TRACE, know WIDEn-N, TRACEn-N, WIDE, TRACE, RELAY and thread them as aliases.)
    5. Alternate keywords that are controlled as alias of “WIDEn-N”
    6. Alternate keywords that are controlled as alias of “TRACEn-N”
    7. Additional rate-limit parameters

Possible way to construct these groups is to have similar style of tag structure as Apache HTTPD does:

```
<digipeater>
   transmit  OH2XYZ-2    # to interface with callsign OH2XYZ-2
   ratelimit 20          # 20 posts per minute
   <trace>
      keys    RELAY,TRACE,WIDE,HEL
      maxreq  4     # Max of requested, default 4
      maxdone 4     # Max of executed, default 4
   </trace>
#  <wide>      # Use internal default
#  </wide>
   <source>
       source OH2XYZ-2       # Repeat what we hear on TX TNC
       filters        ....
       relay-format   digipeated  # default
   </source>
   <source>
       source OH2XYZ-R2      # include auxiliary RX TNC data
       filters        ....
       relay-format   digipeated  # default
   </source>
   <source>
       source OH2XYZ-7       # Repeat what we hear on 70cm
       filters        ....
       relay-format   digipeated   # default
       relay-addlabel 70CM         # Cross-band digi, mark source
   </source>
   <source>
       source DSTAR          # Cross-mode digipeat..
       filters        ....
       relay-format   digipeated   # FIXME: or something else?
       relay-addlabel DSTAR        # Cross-band digi, mark source
       out-path       WIDE2-2
   </source>
   <source>
       source APRSIS         # Tx-IGate some data too!
       filters        ....
       ratelimit      10     # only 10 IGated msgs per minute
       relay-format   third-party # for Tx-IGated
       out-path       WIDE2-2
   </source>
</digipeater>
```

3.3.1 <trace>

Defines a list of keyword prefixes known as “TRACE” keys.

When system has keys to lookup for digipeat processing, it looks first the trace keys, then wide keys. First match is done.

If a per-source trace/wide data is given, they are looked up at first, and only then the global one. Thus per source can override as well as add on global sets.

```
   <trace>
      keys    RELAY,TRACE,WIDE,HEL[1]
      maxreq  4     # Max of requested, default 4
      maxdone 4     # Max of executed, default 4
   </trace>
```

3.3.2 <wide>

Defines a list of keyword prefixes known as “WIDE” keys.

When system has keys to lookup for digipeat processing, it looks first the trace keys, then wide keys. First match is done.

If a per-source trace/wide data is given, they are looked up at first, and only then the global one. Thus per source can override as well as add on global sets.

```
   <wide>
      keys    WIDE,HEL
      maxreq  4     # Max of requested, default 4
      maxdone 4     # Max of executed, default 4
   </wide>
```

1 “HEL” is airport code for Helsinki Airport, so it is quite OK for local area distribution code as well.

3.3.3 <trace>/<wide> Default Rules

The <digipeater> level defaults are:

```
<trace>
   keys    RELAY,TRACE,WIDE
   maxreq  4     # Max of requested, default 4
   maxdone 4     # Max of executed, default 4
</trace>
<wide>
   keys    WIDE  # overridden by <trace>
   maxreq  4     # Max of requested, default 4
   maxdone 4     # Max of executed, default 4
</wide>
```

The <source> level defaults are:

```
<trace>
   keys          # Empty set
   maxreq  0     # Max of requested, undefined
   maxdone 0     # Max of executed, undefined
</trace>
<wide>
   keys          # Empty set
   maxreq  0     # Max of requested, undefined
   maxdone 0     # Max of executed, undefined
</wide>
```

3.4 NetBeacon definitions

Netbeacons are sent only to APRSIS, and *Rfbeacons* to radio transmitters.

```
<netbeacon>
# to      APRSIS              # default for netbeacons
  for     N0CALL-13           # must define
  dest    "APRS"              # must define
  via     "TCPIP,NOGATE"      # optional
  type    "!"                 # optional, default "!"
  symbol  "R&"                # must define
  lat     "6016.30N"          # must define
  lon     "02506.36E"         # must define
  comment "aprx - an Rx-only iGate" # optional
</netbeacon>
```

```
<netbeacon>
# to      APRSIS              # default for netbeacons
  for     N0CALL-13           # must define
  dest    "APRS"              # must define
  via     "TCPIP,NOGATE"      # optional
# Define any APRS message payload in raw format, multiple OK!
  raw     "!6016.35NR02506.36E&aprx - an Rx-only iGate"
  raw     "!6016.35NR02506.36E&aprx - an Rx-only iGate"
</netbeacon>
```

3.5 RfBeacon definitions

Netbeacons are sent only to APRSIS, and *Rfbeacons* to radio transmitters.

```
<rfbeacon>
# to      OH2XYZ-2            # defaults to first transmitter
  for     N0CALL-13           # must define
  dest    "APRS"              # must define
  via     "NOGATE"            # optional
  type    "!"                 # optional, default "!"
  symbol  "R&"                # must define
  lat     "6016.30N"          # must define
  lon     "02506.36E"         # must define
  comment "aprx - an Rx-only iGate" # optional
</rfbeacon>
```

```
<rfbeacon>
# to      OH2XYZ-2            # defaults to first transmitter
  for     OH2XYZ-2            # must define
  dest    "APRS"              # must define
  via     "NOGATE"            # optional
  type    ";"                 # ";" = Object
  name    "OH2XYZ-6"          # object name
  symbol  "R&"                # must define
  lat     "6016.30N"          # must define
  lon     "02506.36E"         # must define
  comment "aprx - an Rx-only iGate" # optional
</rfbeacon>
```

Configuration entry keys are:

name	**Optionality by type**				
	! /	;	)		
to	x(1)	x(1)	x(1)		
for	--	--	--		
dest	--	--	--		
via	x	x	x		
raw	X(2,5)	X(2,5)	X(2,5)		
type	x(2)	x(2)	x(2)		
name	invalid	x(4)	x(4)		
symbol	X(3,4)	X(3,4)	X(3,4)		
lat	X(3,4)	X(3,4)	X(3,4)		
lon	X(3,4)	X(3,4)	X(3,4)		
comment	X(3,4)	X(3,4)	X(3,4)		

Optionality notes:

1. Netbeacons default is APRSIS system, and no transmitter is definable. Rfbeacons default to first transmitter call-sign defined in <interface> sections, any valid transmitter call-sign is OK for “to” keyword.
2. When a “*raw*” is defined, no “*type*” must be defined, nor any other piecewise parts of symbol/item/object definitions.
3. Piecewise definitions of basic positional packets must define at least *type* + *symbol* + *lat* + *lon*. The *comment* is optional, and *name* is rejected if defined.
4. Piecewise definitions of item and object packets must define at least *type* + *name* + *symbol* + *lat* + *lon.* The *comment* is optional.
5. Multiple “*raw*” entries are permitted, they share *to* + *for* + *dest* + *via* -field data, and each generates a beacon entry of its own.
6. Defining timestamped position/object/item packet will get a time-stamp of “h” format (hours, minutes, seconds) generated when beacon is sent. This applies also to *raw* packets! Computer must then have some reliable time source, NTP or GPS.