

APRX Software Requirement Specification

Table of Contents

1	APRX Software Requirement Specification.....	2
1.1	Purpose:.....	2
1.2	Usage Environments:.....	3
2	Treatment rules:.....	4
2.1	Basic IGate rules:.....	4
2.2	Low-Level Transmission Rules:.....	5
2.3	Low-Level Receiving Rules:.....	6
2.4	Additional Tx-IGate rules:.....	6
2.5	Digipeater Rules:.....	7
2.6	Duplicate Detector.....	8
2.7	Radio Interface Statistics Telemetry.....	8
2.8	Individual Call-Signs for Each Receiver, or Not?.....	9
2.9	Beaconing.....	10
2.9.1	Radio Beaconing.....	10
2.9.2	Network beaconing.....	10
3	Configuration Language.....	11
3.1	APRSIS Interface Definition.....	12
3.2	Radio Interface Definitions.....	12
3.3	Digipeating Definitions.....	13
3.3.1	<trace>.....	15
3.3.2	<wide>.....	15
3.3.3	<trace>/<wide> Default Rules.....	16
3.4	Beacon definitions.....	17

5 **1 APRX Software Requirement Specification**

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

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

9 **1.1 Purpose:**

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

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

18

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

21

22

23 1.2 Usage Environments:

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

26 On following one should remember that amateur radio transmitters need a licensed
27 owner/operator/license themselves, but receivers do not:

- 28 1. License-free Receive-Only (RX) IGate, to add more “ears” to hear packets, and to
29 pipe them to APRSIS
- 30 2. Licensed bidirectional IGate, selectively passing messages from radio channels to
31 APRSIS, and from APRSIS to radio channels, but not repeating packets heard on a
32 radio channel back to a radio channel.
- 33 3. Licensed bidirectional IGate plus selectively re-sending of packets heard on radio
34 channels back to radio channels
- 35 4. Licensed system for selectively re-sending of packets heard on radio channels back
36 to other radio channels, and this without bidirectional IGate service.
- 37 5. Licensed system for selectively re-sending of packets heard on radio channels back
38 to radio channels, and doing with with “receive only” IGate, so passing information
39 heard on radio channel to APRSIS, and not the other way at all.

40

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

- 46 1. 3 different frequencies in use, traffic is being relayed in between them, and the
47 APRSIS network.
- 48 2. On each frequency there are multiple receivers, and one well placed transmitter.
- 49 3. Relaying from one frequency to other frequency may end up having different rules,
50 than when re-sending on same frequency: Incoming packet retains traced paths,
51 and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.

52

53

54 2 Treatment rules:

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

58 2.1 Basic IGate rules:

59 General rules for these receiving filters are described here:

60 <http://www.aprs-is.net/IGateDetails.aspx>
61

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

- 63 1. 3rd party packets (data type '}') should have all before and including the data
64 type stripped and then the packet should be processed again starting with
65 step 1 again. There are cases like D-STAR gateway to APRS of D-STAR
66 associated operator (radio) positions.
- 67 2. generic queries (data type '?').
- 68 3. packets with TCPIP, TCPXX, NOGATE, or RFONLY in the header, especially
69 in those opened up from a 3rd party packets.

70
71 Gate message packets and associated posits to RF (Tx-IGate) if

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

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

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

87

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

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

92

93

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

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

99 rx \Rightarrow igate-to-aprsis + digi \Rightarrow tx \Rightarrow rx \Rightarrow igate-to-aprsis + digi (dupe filter stops)

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

102

103 2.2 Low-Level Transmission Rules:

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

108 1. Duplication detector per transmitter: Digipeater and Tx-IGate will ignore packets
109 finding a hit in this subsystem.

110 2. A candidate packet is then subjected to a number of filters, and if approved for it,
111 the packet will be put on duplicate packet detection database (one for each
112 transmitter.) See Digipeater Rules, below.

113 3. Because the system will hear the packets it sends out itself, there must be a global
114 expiring storage for recently sent packets, which the receivers can then compare
115 against. (Around 100 packets of 80-120 bytes each.) This storage gets a full copy
116 of packet being sent out – a full AX.25 frame.

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

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

124

125 2.3 Low-Level Receiving Rules:

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

138 2.4 Additional Tx-IGate rules:

139 The Tx-IGate can have additional rules for control:

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

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

159

160

161 2.5 Digipeater Rules:

162 Digipeater will do following for each transmitter:

- 163 1. Compare candidate packet against duplicate filter, if found, then drop it. (Low-level
164 transmission rules, number 1)
- 165 2. Count number of hops the message has so far done, and...
- 166 3. Figure out the number of hops the message has been requested to do
167 (e.g. "OH2XYZ-1>APRS,OH2RDU*,WIDE7-5: ..." will report that there was request
168 of 7 hops, so far 2 have been executed – one is shown on trace path.)
- 169 4. If either of previous ones are over any of configured limits, the packet is dropped.
- 170 5. FIXME: WIDEn-N/TRACEn-N treatment rules: By default treat both as TRACE,
171 have an option to disable "WIDE-is-TRACE" mode. Possibly additional keywords
172 for cross-band digipeating? (E.g. Bruninga said '6MTRSn-N' would be 'WIDEn-N'
173 on 50 MHz APRS, and only there.)
- 174 6. Multiple filters look inside the message, and can enforce a rule of "repeat only
175 packets within my service area."
- 176 7. FIXME: Cross frequency digipeating? Treat much like Tx-IGate?
177 Relaying from one frequency to other frequency may end up having different rules,
178 than when re-sending on same frequency: Incoming packet retains traced paths,
179 and gets WIDEn-N/TRACEn-N requests replaced with whatever sysop wants.
- 180 8. Cross band relaying may need to add both an indication of "received on 2m", and
181 transmitter identifier: "sent on 6m":
182 "OH2XYZ-1>APRS,RX2M*,OH2RDK-6*,WIDE3-2: ..."
- 183 This "source indication token" may not have anything to do with real receiver
184 identifier, which is always shown on packets passed to APRSIS.
185

186 The MIC-e has a weird way to define same thing as normal packets do with
187 SRCCALL-n>DEST,WIDE2-2: ...

188 The MIC-e way (on specification, practically nobody implements it) is:
189 SRCCALL-n>DEST-2: ...

190

191

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 WIDEn uses SSID to denote number of distribution hops.

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.

225 **2.8 Individual Call-Signs for Each Receiver, or Not?**

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

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

237

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

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

245

2.9 Beacons

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 Beacons

“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.

283 3 Configuration Language

284 System configuration language has several semi-conflicting requirements:

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

291

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

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

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

298

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

301 Some examples:

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

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

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

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

310

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.

```

314 <aprsis>                                # Alternate A, single server, defaults
315     login  OH2XYZ-R1
316     server finland.aprs2.net:14580
317     filter ....
318     heartbeat-timeout 2 minutes
319 </aprsis>
320 <aprsis>                                # Alternate B, multiple alternate servers
321     login  OH2XYZ-R1
322     <server finland.aprs2.net:14580>
323         heartbeat-timeout 2 minutes
324         filter ....
325     </server>
326     <server rotate.aprs.net:14580>
327         heartbeat-timeout 120 seconds
328         filter ....
329         # Alt Login ?
330     </server>
331 </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.

```

335 <interface>
336     serial-device /dev/ttyUSB1 19200 8n1 KISS
337     tx-ok         false           # receive only (default)
338     callsign      OH2XYZ-R2      # KISS subif 0
339 </interface>
340 <interface>
341     serial-device /dev/ttyUSB2 19200 8n1 KISS
342     <kiss-subif 0>
343         callsign OH2XYZ-2
344         tx-ok     true           # This is our transmitter
345     </kiss-subif>
346     <kiss-subif 1>
347         callsign OH2XYZ-R3      # This is receiver
348         tx-ok     false        # receive only (default)
349     </kiss-subif>
350 </interface>
351 <interface>
352     ax25-device OH2XYZ-6
353     tx-ok       true           # This is also transmitter
354 </interface>

```

3.3 Digipeating Definitions

The powerfulness is necessary for controlled digipeating, where traffic from multiple sources gets transmuted 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

374

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

```

377 <digipeater>
378     transmit OH2XYZ-2      # to interface with callsign OH2XYZ-2
379     ratelimit 20           # 20 posts per minute
380     <trace>
381         keys RELAY,TRACE,WIDE,HEL
382         maxreq 4           # Max of requested, default 4
383         maxdone 4          # Max of executed, default 4
384     </trace>
385 # <wide>           # Use internal default
386 # </wide>
387     <source>
388         source OH2XYZ-2      # Repeat what we hear on TX TNC
389         filters ....
390         relay-format digipeated # default
391     </source>
392     <source>
393         source OH2XYZ-R2     # include auxiliary RX TNC data
394         filters ....
395         relay-format digipeated # default
396     </source>
397     <source>
398         source OH2XYZ-7      # Repeat what we hear on 70cm
399         filters ....
400         relay-format digipeated # default
401         relay-addlabel 70CM  # Cross-band digi, mark source
402     </source>
403     <source>
404         source DSTAR         # Cross-mode digipeat..
405         filters ....
406         relay-format digipeated # FIXME: or something else?
407         relay-addlabel DSTAR  # Cross-band digi, mark source
408         out-path WIDE2-2
409     </source>
410     <source>
411         source APRSIS        # Tx-IGate some data too!
412         filters ....
413         ratelimit 10         # only 10 IGated msgs per minute
414         relay-format third-party # for Tx-IGated
415         out-path WIDE2-2
416     </source>
417 </digipeater>

```

418

419 **3.3.1 <trace>**

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

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

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

```
425     <trace>
426         keys      RELAY, TRACE, WIDE, HEL1
427         maxreq    4      # Max of requested, default 4
428         maxdone   4      # Max of executed, default 4
429     </trace>
```

430

431 **3.3.2 <wide>**

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

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

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

```
437     <wide>
438         keys      WIDE, HEL
439         maxreq    4      # Max of requested, default 4
440         maxdone   4      # Max of executed, default 4
441     </wide>
```

442

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

443 3.3.3 <trace>/<wide> Default Rules

444 The <digipeater> level defaults are:

```

445     <trace>
446         keys      RELAY,TRACE,WIDE
447         maxreq    4      # Max of requested, default 4
448         maxdone   4      # Max of executed, default 4
449     </trace>
450     <wide>
451         keys      WIDE   # overridden by <trace>
452         maxreq    4      # Max of requested, default 4
453         maxdone   4      # Max of executed, default 4
454     </wide>

```

455

456 The <source> level defaults are:

```

457     <trace>
458         keys      # Empty set
459         maxreq    0      # Max of requested, undefined
460         maxdone   0      # Max of executed, undefined
461     </trace>
462     <wide>
463         keys      # Empty set
464         maxreq    0      # Max of requested, undefined
465         maxdone   0      # Max of executed, undefined
466     </wide>

```

467

468 3.4 Beacon definitions

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

```

470 <netbeacon>
471 # to      APRSIS          # default for netbeacons
472   for     N0CALL-13       # must define
473   dest    "APRS"         # must define
474   via     "NOGATE"        # optional
475   type    "!"            # optional, default "!"
476   symbol  "R&"           # must define
477   lat     "6016.30N"      # must define
478   lon     "02506.36E"     # must define
479   comment "aprx - an Rx-only iGate" # optional
480 </netbeacon>

481 <netbeacon>
482 # to      APRSIS          # default for netbeacons
483   for     N0CALL-13       # must define
484   dest    "APRS"         # must define
485   via     "NOGATE"        # optional
486 # Define any APRS message payload in raw format:
487   raw     "!6016.35NR02506.36E&aprx - an Rx-only iGate"
488 </netbeacon>

489 <rfbeacon>
490 # to      OH2XYZ-2        # defaults to first transmitter
491   for     N0CALL-13       # must define
492   dest    "APRS"         # must define
493   via     "NOGATE"        # optional
494   type    "!"            # optional, default "!"
495   symbol  "R&"           # must define
496   lat     "6016.30N"      # must define
497   lon     "02506.36E"     # must define
498   comment "aprx - an Rx-only iGate" # optional
499 </rfbeacon>

500 <rfbeacon>
501 # to      OH2XYZ-2        # defaults to first transmitter
502   for     OH2XYZ-2        # must define
503   dest    "APRS"         # must define
504   via     "NOGATE"        # optional
505   type    ";"            # ";" = Object
506   name    "OH2XYZ-6"      # object name
507   symbol  "R&"           # must define
508   lat     "6016.30N"      # must define
509   lon     "02506.36E"     # must define
510   comment "aprx - an Rx-only iGate" # optional
511 </rfbeacon>

```

512