APRStt Implementation Notes

Version 1.1 – August 2014

Introduction

APRStt allows a user, equipped with only DTMF (commonly known as Touch Tone) generation capability, to enter information into the global APRS data network.

You can find more information here: http://www.aprs.org/aprstt.html

This document explains how it was implemented in the Dire Wolf application.

This is a new feature, first added in version 0.7, and is sure to evolve.

Touch Tone Transmission Overview

All transmissions are a sequence of touch tone button presses concluded by the # button. If there are multiple parts, they are separated by the * button. A complete transmission might look like these:

```
B3123456 * C1 * A9A2B42A7A7C71 #
```

You can think of * and # as being like commas and a period in an English sentence.

- * is used only to separate the different parts, which we call "fields" here.
- # means it is the end of a complete group to be processed together.

The first character of each field identifies the type of field:

A = callsign or object characteristics
B = position data
C = comment text or status
D = message text – not defined anywhere, not implemented
0 - **9** = compact all numeric form (macros in this implementation)

All touch tone transmissions must contain a callsign or object name so we can associate the other data with some name.

In most examples, the callsign is shown at the end like this:

```
B3123456 * C1 * A9A2B42A7A7C71 #
```

but that is not a requirement (at least in this implementation). Any other order, such as

```
A9A2B42A7A7C71 * C1 * B3123456 #
```

Is also valid. Typically the callsign is shown at the end because it is convenient to store your callsign and the terminating # in a DTMF memory. You manually press buttons for location and/or status information then send your call from memory.

There are other cases where the opposite order would be more convenient. Suppose you were at a fixed location reporting which runner, bicycle, canoe, parade vehicle, etc. is passing by. In this case, you might manually enter the object number, on your HT keypad, and then send your location and the terminating # from your HT memory.

Any partially accumulated touch tone sequence will be discarded after 5 seconds of no Touch Tone activity. If you make a mistake just wait a while for it to be cleared out and start over.

Modes of Operation

(1) The first is TNC / application server mode.

When a Touch Tone transmission is received, it is placed in the normal APRS packet format and sent to any attached applications for processing. The data type identifier of "t" is used. The rest of the information field is the raw Touch Tone key press data. This is temporary until the official APRS standard has a better way for a dumb TNC, with a DTMF decoder, to convey raw touch tone data to a client application.

Here is an example of what it looks like on the screen in monitoring format:

```
DTMF message
[0] WB2OSZ>GPSAA7:tA9A2B26C7D9D71#
Raw Touch Tone Data, overlayBOX DTMF & RFID & XO w/overlay 7
A9A2B26C7D9D71#
```

The source and destination might contain information useful for troubleshooting but the application should not rely on this. There are no configuration options.

This only gets sent to client applications. It is not sent over the air.

(2) The second is APRStt gateway mode.

Touch Tone transmissions are transformed into regular APRS Object Report format and transmitted as AX.25 frames. If the IGate feature is enabled, they also go directly to an IGate server so we don't rely on some other system to receive them over the air and pass them along.

Most of this document describes which features were implemented, clarifications of ambiguities, rules for composing touch tone sequences, configuration options, and how the tone sequences are interpreted.

Keypad Layout

Letters are assigned to number keys using the same standard arrangement found on modern telephones. Some ham equipment might have different labeling so watch out.

1	2 ABC	3 DEF	A
4 GHI	5 JKL	6 MNO	B
7 PQRS	8 TUV	9 WXYZ	C
*	0 space	#	D

There are two different encodings called:

- **Multi-press** -- Used for comments.

  Letters are represented by one or more presses of the same key depending on their order listed on the button. e.g. Press 5 key once for J, twice for K, thrice for L.

  To specify a digit use the number of letters listed on the button plus one. e.g. Press 5 key four times to get digit 5. When two characters in a row use the same key, use the "A" key as a separator.

- **Two-key** -- Used for callsigns.

  Digits are represented by a single key press.

  Letters (or space) are represented by the corresponding key followed by A, B, C, or D depending on the order of the letter in the order listed.

Examples:

Character	Multi-press	Two-Key	Comments
0	00	0	
1	1	1	No letters on 1 button.
2	2222	2	3 letters -> 4 key presses
9	99999	9	
W	9	9A	
X	99	9B	
Y	999	9C	
Z	9999	9D	
space	0	0A	

There are no punctuation characters other than the space which is handled like the letters. There are no "editing" key sequences. If you make a mistake, wait 5 seconds for the incomplete transmission to be cleared out. If impatient, you could press # before adding the callsign and an invalid transmission will be rejected.

Two converter applications are provided to perform the conversions.

- text2tt – Converts text to both types of encodings.
- tt2text – converts Touch Tone button sequences to text.

Example of usage:

```
$ text2tt abcdefg 0123

Push buttons for multi-press method:
"2A22A2223A33A33340A00122223333"
Push buttons for two-key method:
"2A2B2C3A3B3C4A0A0123"



$ tt2text 2A22A2223A33A33340A00122223333

Could be either type of encoding.
Decoded text from multi-press method:
"ABCDEFG 0123"
Decoded text from two-key method:
"A2A222D3D3334 00122223333"
```

Audible Responses

None at this time.

Callsigns

The following formats are recognized:

A*tt...ttvk* - Full callsign in two key method, numeric overlay, checksum.
A*tt...ttvvk* - Full callsign in two key method, letter overlay, checksum.

A*nnnvk* - "Suffix" abbreviation with 3 digits, numeric overlay, checksum.
A*nnnvvk* - "Suffix" abbreviation with 3 digits, letter overlay, checksum.

A*nnn* - "Suffix" abbreviation with 3 digits. No overlay. No checksum.

(Not sure yet about the "spelled suffixes.")

A "suffix" abbreviation / overlay combination will be replaced by the corresponding full callsign if found in memory of recent activity.

Object Names & Symbols

This is an implementation-specific extension to the "standard." Even if this is never sent over the air, it is still very useful combined with macros described later.

These new formats overcome several shortcomings in the standard:

- It is possible to enter 9 character object names, not just 6 character identifiers (callsigns).
- Checksums are not required. Imagine the difficulty in calculating the identifier checksum for each bicycle whizzing by in a race!
- Symbols, other than a box with an overlay character, can be specified.

Notice how a callsign or abbreviation touch tone sequence always has a digit after the initial "A." This leaves open the opportunity to define other formats that have A, B, C, or D after the initial A.

Dire Wolf adds these unique extensions:

AA*tt...* - Object name, two key method, up to 9 characters. Object name may contain letters, digits, and space. No checksum.

AB1*nn* - Symbol from primary symbol table. Two digits *nn* are the same as in the **GPSC***nn* generic address used as a destination.

AB2*nn* - Symbol from alternate symbol table. Two digits *nn* are the same as in the **GPSE***nn* generic address used as a destination.

AB0*nnvv* - Symbol from alternate symbol table. Two digits *nn* are the same as in the **GPSE***nn* generic address used as a destination. *vv* is an overlay digit or letter in two key method.

AC... - Possible future use for other object properties.

AD... - Possible future use for other object properties.

Locations

APRStt literature lists a wide variety of location formats which are still evolving. Early 2013, we found this in the specification:

```
B0x*         One of 10 special positions
B1xy*        1 digit XY  (   10 mi in 60 mi area)  (default)
                         (or  1 mi in 10 mi area)
                         (or .1 mi in  1 mi area)
B2xxyy*      2 digit XY  (    1 mi in 60 mi area)  (default)
                         (or .1 mi in 10 mi area)
                         (or 60 ft in  1 mi area)
B3xxxyyy*    3 digit XY  (   .1 mi in 60 mi area)  (default)
                         (or 60 ft in 10 mi area)
B4xxxxyyyy*  4 digit XY  (   60 ft in 60 mi area)  (default)

B5zzzmm*     at bearing zzz range mm miles
B6EEENNN*    SAR UTM Grid - Easting and Northing
B7RRRMMM*    Road RRR, Milemark MMM
B8haaaoooo*  Space Format (hemisphere, MSB's of lAt and lOng
B9...    *   Table Interpolation.  Example B9nn* for a list of
             100 named locations at Jamboree, then nn digits
             can specify any of those 99 locations
```

The Jamboree 2013 APRStt literature instructed people to use the **B***yyyxxx* format. This doesn't correspond to any of the above which always have a fixed format identifier digit after the B. In August, the spec was changed so that locations use the Y X (latitude, longitude) order rather than the previous X Y order. Except for UTM coordinates which are always X (easting) then Y (northing).

This implementation generalizes most of them into four very flexible types:

- Point – a specific location.
- Vector – bearing and distance from a specified point.
- Grid – a rectangular area, based on latitude and longitude.
- UTM – a rectangular area, based on distances in meters.

Locations – Point

The more general **point** type implements these 3 standard types

- B0… ten positions
- B7… route / mile mark
- B9… hundred named locations

The configuration file format looks like this:

TTPOINT **B***n…* *latitude* *longitude*

Where, *n…* is one or more digits.

In each case, the latitude and longitude can be listed as signed decimal degrees (negative for south or west) or in degrees / minute / hemisphere format. The degree symbol is not part of ASCII so ^ can be used instead.

Examples:

```
TTPOINT  B01   37^55.37N  81^7.86W              -- special position 1 of 10
TTPOINT  B7495088  42.605237  -71.34456         -- route 495, mile mark 88
TTPOINT  B934  42.605237  -71.34456             -- location 34 out of 100
```

If the received data was "B934", it would simply look for an exact match among the points listed.

Locations – Vector

The **vector** type has a starting point, bearing, and distance. Configuration file format:

TTVECTOR **B**5bbbddd… *latitude* *longitude* *scale* *unit*

Where, 5 must match the tone received after the B.
bbb is a place holder for 3 digit bearing in degrees, clockwise from north.
ddd… is a place holder for distance, at least 1 digit.
Scale is a multiplier to apply to the received digits. This allows us to have fractions. For example distance of "1234" and a scale of 0.01 would represent 12.34 km or miles.
Unit is km, mile, or other common unit.

Example: Configuration file: for Hilltop Tower center. Exactly 3 digits are required for the bearing. In this case the distance is also 3 digits.

```
TTVECTOR  B5bbbddd  37^55.37N  81^7.86W  0.01  mi
```

Received data:

```
B5206070
```

This means 0.70 mile in the direction of 206 degrees (SSW). It should end up at the Archery & Target Range.

Locations – Grid

The rectangular **grid** format has a variable number digits for latitude (y) and longitude (x). Each configuration file item can have optional fixed digits that must match and x and y characters for the coordinate positions.

Coordinates define the edges of the box area.

Latitude for minimum value (y… = all zeros).
Longitude for minimum value (x… = all zeros).
Latitude for maximum value (y… = all nines).
Longitude for maximum value (x… = all nines).

They can be any arbitrary locations but they correspond to fractional digits in these examples.

```
TTGRID   B1xy       12.0  34.0   12.9     34.9
TTGRID   B2xxyy     12.0  34.0   12.99    34.99
TTGRID   B3xxxyyy   12.0  34.0   12.999   34.999
TTGRID   B4xxxxyyyy 12.0  34.0   12.9999  34.9999
TTGRID   Byyyxxx    37^50.00N  81^00.00W  37^59.99N  81^09.99W
```

Examples of received tones and resulting latitude and longitude:

B100 → 12.0 34.0
B101 → 12.0 34.1
B102 → 12.0 34.2
B109 → 12.0 34.9
B189 → 12.8 34.9
B199 → 12.9 34.9

The `Byyyxxx` example is the format mentioned in http://www.aprs.org/aprs-jamboree-2013.html, version of mid February 2013. Note that the x, y order is reversed from the others. It's all handled by the same general code that treats the y digit positions as latitude and x positions as longitude.

The received touch tone sequence B533686 would be translated to 37°5**5**.**33**' N 81°0**6**.**86**' W.

Do you want to send coordinates in the X Y order or Y X order? This implementation doesn't care. No coding changes are required. Just change one line of the configuration file. You can even do bizarre things like interleaving the coordinates (e.g. B2xyxy) but it's probably not a sensible thing to do.

Positions – UTM

UTM coordinates use distances in meters rather than angles in degrees. The configuration file items have this format:

TTUTM **B**6xxx…yyy… *Zone [Scale [X-offset Y-offset]]*

Where, 6 must match the first digit sent after B.
xxx… is a placeholder for up to 6 "easting" (X coordinate) digits
yyy… is a placeholder for up to 7 "northing" (Y coordinate) digits.

Zone is the UTM zone and optional latitude band.
Scale is a multiplier to apply to the received digits. This allows us to drop trailing digits for less resolution.
X-offset & Y-offset are added to the received data so leading digits can be omitted from the transmission.

How do we know if the coordinates are in the northern or southern hemisphere? A zone with only a number is assumed to be northern hemisphere. It can also be suffixed with a longitudinal band of N, P, Q, R, S, T, U, V, W, or X. It doesn't matter which one because the Y coordinate is relative to the equator, not the band. For the southern hemisphere, a suffix of C, D, E, F, G, H, J, K, L, or M must be used. Again, it doesn't matter which one because the Y coordinate is relative to 10,000 km south of the equator.

The simplest configuration file format would need room for 6 digits of "easting" (X) coordinate and 7 digits for the "northing" (Y) coordinate.

```
TTUTM  B6xxxxxxyyyyyyy  19
```

Sample received data:

```
B63075094721178
```

That's a lot of digits to enter. If your application doesn't need resolution of a meter, you can drop the last digit of each coordinate and specify a scaling factor for the transmitted string of digits. For example, to get 10 meter resolution we can use only 5 and 6 digits with a scale factor of 10:

```
TTUTM  B6xxxxxyyyyyy  19  10
```

That's still pretty long. In many cases, the region of interest will not be that large so it is feasible to use a smaller number of digits. For example, when searching a forest for a lost person, it might be possible to express the entire region in a form like this:

```
30xxx0  472yyy0
```

The xxx and yyy ranges would extend over a 10 x 10 km area with 10 meter resolution. Use a configuration like this:

```
TTUTM  B6xxxyyy  19  10  300000  4720000
```

Transmitted data can now be much more compact. E.g.

```
B6613601
```

This will get transformed into 30**613**0 472**601**0

Notice that a received string could match multiple patterns. Does the received B533686 match pattern Byyyxxx (location on grid) or B5bbbdd (bearing and 2digit distance)? The patterns are tested in the order defined and the first match wins.

Two utilities, **ll2utm** and **utm2ll**, are included to convert between Latitude / Longitude and UTM coordinates.

Examples:

```
$ ll2utm 42.662139 -71.365553
zone = 19T, easting = 306130, northing = 4726010

$ utm2ll 19T 306130 4726010
latitude = 42.662139, longitude = -71.365553
```

Comment / Status

This implementation recognizes all standard types:

C*n*	- Exactly one digit indicates a predefined comment. 0 = (none, default) 1 = /off duty 2 = /enroute 3 = /in service 4 = /returning 5 = /committed 6 = /special 7 = /priority 8 = /emergency 9 = /custom 1
C*nnnnnn*	- Exactly 6 digits are a frequency.
C*ttt…ttt*	- Anything else is general text in multi-press encoding.

Compact all numeric form (macros)

Pressing all those buttons can get pretty tedious and error prone. Suppose you wanted to use APRStt to report positions of runners, bicycles, boats, or parade vehicles along some route. You might send a sequence something like this to report that bicycle 123 is near predefined position 78 along the route; the rider is injured and needs medical attention.

```
C8 * B978 * AB166 * AA2B4C5B3B0A123 #
```

C8 = predefined "emergency" comment
B978 = standard form for one of 100 defined locations.
AB166 = primary symbol table, bicycle.
AA… = object name "BIKE 123"

Try entering that on your HT keypad correctly as bicycles go whizzing by! There has been some discussion about a very compact form that could be used for situations like this. It was also desirable that the A, B, C, and D buttons would not be required because some radios do not have them or can't store them in DTMF memories. This lead to discussions of a "runner mode" with short touch tone sequences like this:

bbnnn…#

where,

bb is a 2 digit location.
nnn is the "runner" number with a configurable number of digits.

Rather than hard-coding numerous special cases for every new situation, a more flexible, and simple, approach has been taken. The system operator can define new formats rather changing the source code.

Macros allow you to define very short transmission formats and their longer equivalent.

The TTMACRO configuration option is used to map compact **all numeric** fields into the longer standard form before processing. The general form is:

TTMACRO *x…y…z…* *Touch tone sequence with x, y, and z for substitutions.*

Where, *x…y…z… are specific digits that must match and/or the lower case letters x, y, or z as placeholders for separating a received digit sequence into fixed length pieces.*

This should be easier to understand with a couple examples.

Configuration file: These are the actual characters, not some meta representation.

```
TTMACRO   xxyyy   B9xx*AB166*AA2B4C5B3B0Ayyy
```

Here we are saying that when we receive a touch tone sequence of 5 digits, followed by #, of course, this rule will be applied. Take the first two digits and remember them as x. Take the other 3 digits and remember then as y. Substitute the received digits into the x and y positions in the definition.

To report bike 123 at location 78, simply press these buttons: 78123#.

There are five digits so it would match the macro pattern for five digits. xx would be 78 and yyy would be 123.

Original pattern: `B9xx*AB166*AA2B4C5B3B0Ayyy`
After substitution: `B9`**`78`**`*AB166*AA2B4C5B3B0A`**`123`**

This expanded form would not be visible outside. It is not passed along to an attached client application. It is just used internally. It is processed as if it had been heard over the air and converted to an APRS Object and transmitted. The object name would be "BIKE 123" and the location would be whatever was defined with "TTPOINT B978 …"

Suppose you also wanted the ability to attach an optional status to the object. You could define a rule on how to process a sequence with exactly 6 digits.

```
TTMACRO   xxyyyz   Cz*B9xx*AB166*AA2B4C5B3B0Ayyy
```

Here we are saying that when we receive a touch tone sequence of 6 digits, always terminated by #, of course, this rule will be applied. Take the first two digits and remember them in the x variable. Take the next 3 digits and remember them as y. Remember the final digit as z. Substitute the received digits into the x, y, and z positions in the defintion. If we were to receive 781239#, xx would be 78, yyy would be 123, and z would be 9.

Original pattern: `Cz*B9xx*AB166*AA2B4C5B3B0Ayyy`
After substitution: `C`**`9`**`*B9`**`78`**`*AB166*AA2B4C5B3B0A`**`123`**

Status would be set to "Custom 1."

Alternatively, you might define a single digit macro to generate the status. This would be less error prone.

```
TTMACRO   z   Cz
```

The transmitted touch tone sequence would then be:

```
9*78123#
```

This is first separated, at the *, into two fields of "9" and "78123". The field with one digit is expanded by the macro rule for one digit. The field with five digits expanded as before. Again, we end up with

`C`**`9`**`*B9`**`78`**`*AB166*AA2B4C5B3B0A`**`123`**

which is processed as if someone had typed that all in manually.

Suppose there were multiple types of objects to track. It would be nice to have different name prefixes and even display icons to easily distinguish them.

Object numbers 100 – 199 = bicycle
Object numbers 200 – 299 = fire truck
Others = dog

Define these 3 rules:

```
TTMACRO   xx1yy   B9xx*AB166*AA2B4C5B3B0A1yy
TTMACRO   xx2yy   B9xx*AB170*AA3C4C7C3B0A2yy
TTMACRO   xxyyy   B9xx*AB180*AA3A6C4A0Ayyy
```

The touch tone sequence 78**1**23# would match the first one because it requires 1 in the middle position.

The touch tone sequence 78**2**23# would match the second one because it requires 2 in the middle position. The object name is "FIRE 223" and the fire truck icon is used.

The touch tone sequence 78**3**23# would match the third one because y in the middle position matches anything and the earlier patterns did not catch it. The object name is "DOG 323" and the puppy dog icon shows up on the map.

Traditional forms and macros can be combined. For example,

```
C3*C146520*78223#
```

means the fire truck is "in service" and listening on 146.52 MHz.

Punch this in on the old keypad:

```
9*01123#
C3*C146520*02223#
03323#
```

The troubleshooting output illustrates the transformation process.

```
DTMF message
[0.dtmf] APRSTT>GPSE33:t9*01123#
Raw Touch Tone Data, numbered box
9*01123#
Macro tone sequence: '9'
Matched pattern  13: 'z', x=, y=, z=9, b=, d=
Replace with:        'Cz'
After substitution:  'C9'
Macro tone sequence: '01123'
Matched pattern  10: 'xx1yy', x=01, y=23, z=, b=, d=
Replace with:        'B9xx*AB166*AA2B4C5B3B0A1yy'
After substitution:  'B901*AB166*AA2B4C5B3B0A123'
[0L] WB2OSZ-5>APDW10:;BIKE 123 *221245z4239.68N/07121.87Wb /custom 1  !T01!
```

```
DTMF message
[0.dtmf] APRSTT>GPSE33:tC3*C146520*02223#
Raw Touch Tone Data, numbered box
C3*C146520*02223#
Macro tone sequence: '02223'
Matched pattern  11: 'xx2yy', x=02, y=23, z=, b=, d=
Replace with:        'B9xx*AB170*AA3C4C7C3B0A2yy'
After substitution:  'B902*AB170*AA3C4C7C3B0A223'
[0L] WB2OSZ-5>APDW10:;FIRE 223 *222330z4239.62N/07121.87Wf146.520MHz /in service!T02!
```

```
DTMF message
[0.dtmf] APRSTT>GPSE33:t03323#
Raw Touch Tone Data, numbered box
03323#
Macro tone sequence: '03323'
Matched pattern  12: 'xxyyy', x=03, y=323, z=, b=, d=
Replace with:        'B9xx*AB180*AA3A6C4A0Ayyy'
After substitution:  'B903*AB180*AA3A6C4A0A323'
[0L] WB2OSZ-5>APDW10:;DOG 323  *221304z4239.54N/07121.87Wp !T03!
```

The "corral"

APRStt users might not report their position. In this case, we only know they are in range of the receiver. How do we represent their location? How can they be positioned on a map?

The traditional approach is to assign them arbitrary locations in the "corral." Some implementations always place this next to the gateway location. This implementation is a little more flexible. The corral can be positioned in some other sparsely populated location on the map.

```
TTCORRAL    latitude    longitude     offset
```

In the first example below, the list starts at the top and grows downward. In the second example, we start at the bottom and go up from there. In each case the spacing is 0.02 minute.

```
TTCORRAL   37^56.00N  81^7.00W  0^0.02S
TTCORRAL   37^55.50N  81^7.00W  0^0.02N
```

This has a couple unfortunate consequences. It gives the illusion that we know where the station is located. It might be obvious when displayed on a map, but a text only display, built in to transceiver, doesn't make it clear. A suitable offset also depends on the map display scale. If zoomed out too far, the stations will be piled on top of each other and unreadable.

Enhanced position reporting ?

This behavior is a good approximation and backward compatible with existing systems, but it has some weaknesses. In some cases, the conversion loses information. In some cases, the conversion supplies a made up location.

When looking at a map, you can make a pretty good guess what is in the “corral” due to an unknown location. However, it is not clear to a station with a text only display such as TM-D710A. Actual positions, and those assigned by the APRStt gateway look the same.

The APRStt gateway operator needs to set corral parameters such as starting location and spacing. A good spacing for one display might not be so good for a different display size or zoom level. An APRStt-aware application might want to want to use a different location for the corral but it doesn't have a good way of telling whether the user provided a position or if the Gateway supplied it.

Another case is now being discussed where multiple objects might be at the same location. Besides tracking "runners" we might also use APRStt to track the movement of medical staff and special equipment at first aid stations. If each of the objects was reported with the same location (e.g. B925 for first aid station 25), they would all be piled on top of each other on a map. If an APRStt-aware application knew they were all at predefined location 25, it could perform its own "corral" function to display them in a non-overlapping way.

The intention of APRS is to be a real time tactical information system, not just a bunch of icons on a map. An application might want to keep track of what people and special equipment and are at each location. This is difficult to do when everything gets boiled down to just a latitude and longitude.

I would like to propose a simple extension to retain more information for possible use by applications. The idea is to add something indicating whether the location is unknown or one of the predefined locations. This should be human readable so someone with a text-only display can instantly see that an object is at first aid station 25 or the location is unknown (the corral).

Rather than making up something new, the "!DAO!" option already exists to add enhanced information about the lat/lon position. The gateway would add another 5 characters to the end of the comment to provide more clarification about how the location was derived. The formats might be:

`!T  !`	The position is unknown. (the "corral" lat/lon were assigned by the gateway.)
`!T`*`n`*` !`	The position is one of ten predefined locations (i.e. the B0n tone sequence) (where 'n' represents some digit 0-9.)
`!T`*`nn`*`!`	The position is one of 100 predefined locations (i.e. the B9nn tone sequence)
`!TB`*`n`*`!`	The position was specified by some other method. n is the number following

B in the position. For example, “!TB6!” indicates UTM coordinates where used. A display application might want to use this to provide UTM coordinates instead of or in addition to the latitude and longitude.

Advantages:

- It is simple.
- It is uses an existing part of the protocol specification rather than making up something new.
- It is backward compatible. Most applications would just ignore these characters in the comment.
- **It is human readable.** Someone with a text only display would recognize "!T25!" as being location 25, e.g. first aid station 25. If you saw "!T !" you would know the actual location is unknown and not to look for someone at the object coordinates.

An application, capable of recognizing this could use the information in a couple different ways.

It could override the object lat/lon and perform its own corralling based on the display size and zoom level. One region for unknown locations ("!T "). Possibly other regions for checkpoints ("!T25!").

This also gets back to the principle that APRS is a real time tactical awareness tool, not just icons on a map. Someone might want to know who is at first aid station 25. A suitable application could easily display a list of objects that had "!T25!" in the comment.

Object Report Format

The object header format is represented in the configuration file with the TTOBJ command. It contains the radio channel for transmission (0 is first or only) and the packet header.

```
TTOBJ  0  WB2OSZ-5>APDW07,WIDE1-1
```

It shows up on the screen something like this:

```
[0L] WB2OSZ-5>APDW07:;WB2OSZ-12*250151z4236.31N707120.67WA /off duty
```

This might change. We already know the station name from MYCALL. The software version is known. No sense in forcing people to enter it again.

This is the configuration option that actually enables the gateway. In earlier versions, the DTMF decoder was always active because it took a negligible amount of CPU time. Unfortunately this sometimes resulted in too many false positives from some other types of digital transmissions heard on HF.

Starting in version 1.0, the DTMF decoder is enabled only when the APRStt gateway is configured.

Beaconing

There is nothing special for APRStt. Announce the gateway with the same technique you would use to advertize a digipeater or other station with a fixed location. Example:

```
OBEACON  DELAY=0:15  EVERY=10:00  VIA=WIDE1-1
      OBJNAME=WB2OSZ-tt  SYMBOL=APRStt
      LAT=42^37.14N  LONG=71^20.83W
      COMMENT="APRStt Gateway"
```

This should all be on a single line in the configuration file. It is shown as multiple lines here due to page width limitations.