APRStt Implementation Notes

Version 1.3 – DEVELOPMENT snapshot D -- July 2015

This is a development snapshot. Some features might be partially completed and/or not tested.
Everything is subject to change. Your feedback is welcome.

Very few hams have portable equipment for APRS but nearly everyone has a handheld radio that can send DTMF tones. 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

It's a great idea. It's been around since the turn of the century but never caught on. Earlier implementations required specialized hardware and were not easily adaptable to new situations.

This document explains how the APRStt concept was implemented in the Dire Wolf application. No special hardware is required. Audio, from the receiver, is captured by the computer's soundcard. DTMF decoding is all in software. This will run on Windows and various flavors of Linux including single board computers such as the Raspberry Pi, Beaglebone, cubieboard2, etc. Configuration options are very flexible so it can be adapted to a wide variety of different situations.

CONTENTS

1 A Brief History of APRStt 3
2 Modes of Operation 5
2.1 Traditional APRStt gateway 5
2.2 APRStt front end for Applications 5
2.3 DTMF to speech application framework 5
3 Touch Tone Transmission Overview 7
3.1 Standard Keypad Layout 8
3.2 Alternate Keypad Layout 9
3.3 Conversion Utilities 10
4 Callsigns & Object Names 12
4.1 Traditional callsigns 12
4.2 Object Names & Symbols 12
5 Locations 14
5.1 Point 14
5.2 Vector 15
5.3 Grid 16
5.4 UTM 16
5.5 USNG & MGRS 18
5.6 Maidenhead Grid Square Locator 19
5.7 QIKcom-2 Satellite Grid Squares 20
6 Comment / Status 21
7 Compact all numeric form (macros) 22
7.1 Text to Tone Sequence Conversions 26
8 Audible Responses 28
8.1 Invoking your own application. 29
9 Object Report Generation and Routing 31
9.1 The “corral” 32
9.2 Enhanced position reporting 32
9.3 Location Ambiguity 34

1 A Brief History of APRStt

Let's first take a step back and quickly review the evolution of APRStt implementations.

2002 - APRStt for DOS

Ran on a DOS PC faster than 20 MHz. Used a DTMF decoder chip attached to the parallel printer port.

ftp://ftp.tapr.org/aprssig/dosstuff/APRSdos/aprstt.txt

2009 - CSS Radio Spotter

There are several mentions of its planned debut at Dayton 2009 but no mention of it after that.

http://www.eham.net/articles/21601

2010 - APRSspeak

One radio is connected to the soundcard, the DTMF decoding was done in software, and responses were in voice. The second radio is connected to an external TNC attached by serial port. Vanished since that time.

https://www.tapr.org/pdf/DCC2010-APRSSpeak-KA2UPW.pdf

2011 – APRS Touch Tone Python Scripts

Open source version written for LInux and using "festival" for speech. "Proof-of - concept" not updated since 2011.

https://github.com/sshamilton/APRStt

2012 (?) - Tiny Trak4

Adding an optional DTMF decoder to this popular Tracker / Digipeater / TNC turns it into an APRStt gateway

http://www.byonics.com/tinytrak4/

2013 - Dire Wolf, version 0.7

A DTMF decoder / APRStt translator is integrated with a "soundcard" TNC, APRS digipeater, and IGate. Very flexible configuration can accommodate traditional and user customizable tone sequences. Major deficiency is lack of audible response so you don't know if tone sequence was received properly.

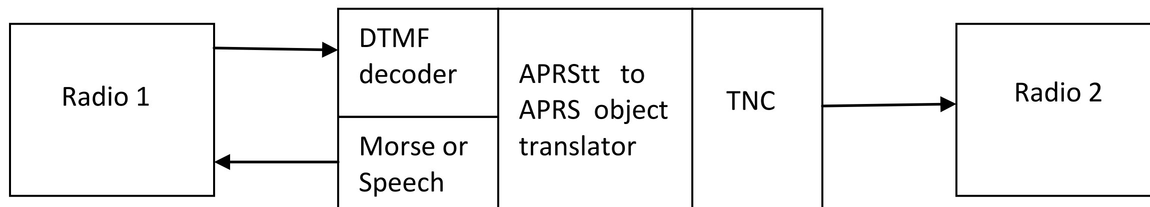
2015 - Dire Wolf, version 1.3

Tone sequences now produce responses with synthesized speech and/or Morse code. Additional options allow more types of positions, callsign/object representations, custom voice responses, and more. Results from the APRStt translator can be sent to any combination of a transmitter, Internet Gateway, or attached applications. See related document, **APRStt-interface-for-SARTrack.pdf**, for a complete example of how it could be used.

2 Modes of Operation

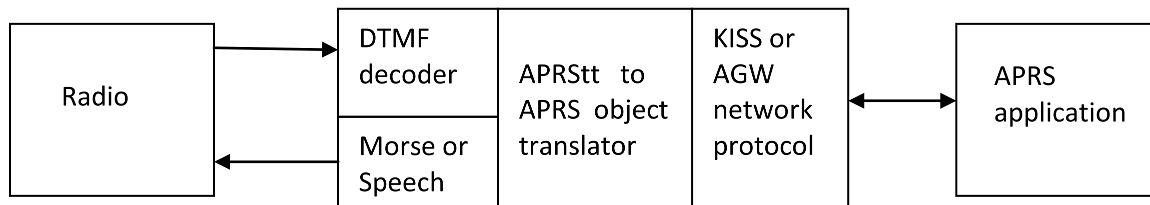
2.1 Traditional APRStt gateway

In the traditional model, users send tone sequences with combinations of a callsign (or other object name), location, and status/comment using the Touch Tone keypad. An audible response, using Morse Code or speech, indicates whether the tone sequence was valid. This information is translated into an APRS "object report" and transmitted, usually on a different radio channel.



2.2 APRStt front end for Applications

Rather than transmitting the APRS "object report" over the radio, it is sent directly to an attached application such as SARTrack, Xastir, APRSISCE/32, or YAAC. These applications can respond with Morse Code, synthesized voice, or APRS packets to the channel where the tones were heard and/or additional radios.



2.3 DTMF to speech application framework

In this case, we eliminate the APRStt translator, in the illustration above, and the user-supplied application is responsible for interpreting the tone sequences.

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.

This only gets sent to client applications. It is not sent over the air. The sample application "**ttcalc**" provides an example of how you might develop your own application to process the DTMF tone sequences and send a reply. Replies can be as AX.25 frames, Morse Code, or speech on multiple radio channels.

3 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 this:

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

You can think of * and # as being like commas and a period in a 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 button press of each field identifies the type of field:

A = callsign or object characteristics
B = location 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 many examples, the callsign is shown at the end like this:

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

This is not a requirement. Other orders are also valid.

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

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 information, 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.

How do we go about sending letters with a numeric keypad? There are multiple encoding methods used in different situations.

3.1 Standard Keypad Layout

Letters are assigned to number keys using the same standard arrangement found on modern telephones.

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 three 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.

  This encoding allows the mixing of letters and digits.

- **Two-digit** -- Used for Maidenhead Grid Locators.

  Digits are represented by a single key press.

  Letters are represented by the corresponding key (2 through 9) followed by 1, 2, 3, or 4 depending on the order of the letter in the order listed.

You are probably wondering: How would we distinguish between the letter W or the digits 9 1. It depends on the character position. We will explain in a later section.

Examples:

Character	Multi-press	Two-Key	Two-Digit	Comments
0	00	0	0	
1	1	1	1	No letters on 1 button.
2	2222	2	2	3 letters -> 4 key presses for digit
9	99999	9	9	
W	9	9A	91	
X	99	9B	92	
Y	999	9C	93	
Z	9999	9D	94	
space	0	0A	n/a	

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.

3.2 Alternate Keypad Layout

The QIKcom-2 APRStt satellite project (http://aprs.org/qikcom-2.html) uses a different encoding for callsigns. This is based on an older keypad layout where Q and Z were on the 1 button if present at all.

1 QZ	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PRS	8 TUV	9 WXY
*	0 space	#

Callsigns are encoded as 6 digits corresponding to the buttons of the callsign characters. An additional 4 digits specify which of the possible characters are used in each case.

This is best explained by example.

```
W  B  4  A  P  R       Callsign.
9  2  4  2  7  7       Button for each character above.
1  2  0  1  1  2       Position for each character.
                           0 for the digit.
                           1 for first letter, 2 for second letter, etc.
                           Space is like first letter for the 0 button.
```

The bottom line is treated as a base 4 number. When converted to base 10, it becomes 1558. The final result would be 9242771558. For callsigns shorter than 6 characters, we append spaces which are represented by “0” for the button and 1 in the base 4 number because it is like the first letter for the button.

3.3 Conversion Utilities

All of these different types of encoding are confusing and some are not so easy to perform manually. Two converter applications are provided to perform the conversions.

- **text2tt** – Converts text to various types of encodings sent over the air.
- **tt2text** – converts Touch Tone button sequences, sent over the air, to ordinary text.

Each evaluates the different possible conversions for the given values and prints the results for any types of conversions that would be valid for the given data.

Examples of usage:

```
$ text2tt abcdefg 0123

Push buttons for multi-press method:
"2A22A2223A33A33340A00122223333"    checksum for call = 5
Push buttons for two-key method:
"2A2B2C3A3B3C4A0A0123"    checksum for call = 1

$ tt2text 2A22A2223A33A33340A00122223333

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

$ text2tt wb4apr

Push buttons for multi-press method:
"922444427A777"    checksum for call = 9
Push buttons for two-key method:
"9A2B42A7A7C"    checksum for call = 4
Push buttons for fixed length 10 digit method:
"9242771558"
```

```
$ tt2text 9242771558

Could be either type of encoding.
Decoded text from multi-press method:
"WAGAQ1KT"
Decoded text from two-key method:
"9242771558"
Decoded callsign from 10 digit method:
"WB4APR"

$ text2tt EM29QE78

Push buttons for multi-press method:
"33622229999977337777778888"    checksum for call = 2
Push buttons for two-key method:
"3B6A297B3B78"    checksum for call = 8
Push buttons for Maidenhead Grid Square Locator:
"326129723278"

$ tt2text 326129723278

Could be either type of encoding.
Decoded text from multi-press method:
"DAM1AWPADAPT"
Decoded text from two-key method:
"326129723278"
Decoded Maidenhead Locator from DTMF digits:
"EM29QE78"
```

4 Callsigns & Object Names

Each transmission must include a callsign or object name which boil down to the same thing.

4.1 Traditional callsigns

The following traditional formats are recognized. Upper case "A" means literally the "A" button. Lower case letters are placeholders for digits. "…" indicates it is variable length.

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.

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

4.2 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 the traditional callsign formats always have 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*nnnnnnnnnn* - Callsign represented by exactly 10 digits as defined in earlier section. This is the encoding used by the QIKcom-2 satellite.

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

The symbol codes can be found in Appendix 2 of the **APRS Protocol Reference**.
http://www.aprs.org/doc/APRS101.PDF You can also get a list of them by running direwolf with the "-S" (upper case S) command line option.

5 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. This makes more sense because we usually use Latitude then Longitude order. UTM coordinates are always X (easting) then Y (northing).

Trying to keep up with all of these variations would be quite a chore. Some are not very flexible. For instance the "bearing" style has a resolution of 1 mile. Rather than being limited to a small number of fixed formats, you can define your own in the configuration file.

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.

You can also specify locations based on GMRS, USNG or Maidenhead Locators.

5.1 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   Bn…   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 ^ is 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.

5.2 Vector

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

```
TTVECTOR   B5bbbddd…   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.

5.3 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.

5.4 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 to indicate hemisphere.

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 latitudinal 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 43.775 11.25896
UTM zone = 32, hemisphere = N, easting = 681795, northing = 4849363
MGRS =  32TPP85   32TPP8149   32TPP818494   32TPP81804936   32TPP8179549363
USNG =  32TPP84   32TPP8049   32TPP817493   32TPP81794936   32TPP8179549363

$ utm2ll 32 681795 4849363
from UTM, latitude = 43.774999, longitude = 11.258957

$ utm2ll 32TPP81794936
from USNG, latitude = 43.774974, longitude = 11.258894
from MGRS, latitude = 43.774974, longitude = 11.258894
```

There are numerous on-line and downloadable coordinate converters available. You know how to use Google to find them. If using Debian/Ubuntu/Raspbian, you can install one with:

```
sudo apt-get install geotranz
```

Dire Wolf is using some of the conversion functions from this package. *"The product was developed using GEOTRANS, a product of the National Geospatial-Intelligence Agency (NGA) and U.S. Army Engineering Research and Development Center."* http://earth-info.nga.mil/GandG/geotrans/index.html

5.5 USNG & MGRS

These a different representation of UTM coordinates. USNG & MGRS are essentially the same but results might differ by about a meter due to the slightly different mathematical conversion models used. To reduce the number of digits that need to be sent, the zones are broken into 100 km squares represented by two letters. Finally we have a variable number of digits depending on precision requirements. The same number of digits (same precision) must be used for both easting and northing.

32T	PP	8179	4936
Grid zone	100 km Square	easting (right)	northing (up)

For more details, see http://www.fgdc.gov/usng

The same location could be represented by any of these depending on the desired precision:

32T PP	somewhere in 100 km x 100 km square
32T PP 8 4	10 km x 10 km square
32T PP 80 49	1 km x 1 km square
32T PP 817 493	100 m x 100 m square
32T PP 8179 4936	10 m x 10 m square – MOST COMMON
32T PP 81795 49363	1 meter x 1 meter square

Notice how the low order digits are truncated. There is no rounding when reducing to a smaller number of digits. 49 is truncated to 4 rather than rounded up to 5. Truncating the lower digits means that the resulting location is the lower left (south west) corner of the region. The number of digits implies the size.

The configuration file items have this format:

```
TTUSNG   B[n]xxx…yyy…   zone_square
TTMGRS   B[n]xxx…yyy…   zone_square
```

Where, *n* is an optional digit which must match the first digit sent after B. 6 is suggested.
xxx… is a placeholder to match 1 to 5 "easting" (X coordinate) digits
yyy… is a placeholder for "northing" (Y coordinate) digits. Must be same length as *xxx…*
zone_square is the zone and two letter square.

Configuration file example:

```
TTUSNG   Bxxxxyyyy   32TPP
```

Notice that we will match "B" followed by exactly 8 digits. Sample received data:

```
B81794936
```

The zone & square, in the configuration, is combined with the received digits and we end up with `32TPP81794936`.

5.6 Maidenhead Grid Square Locator

Normally Maidenhead Grid Square Locators are written as alternating pairs of letters and pairs of digits to the desired resolution. When sent by tones, the letters are replaced by pairs of digits as explained in an earlier section.

The format must be the "B" button, optionally some other button, and some number of lower case x characters to match the digits which will be sent over the radio.

`TTMHEAD    BAxxxxxx   [ ` *prefix]*

The optional prefix, of 10, 6 or 4 digits, is prepended to the received dgits and the result is converted from digits to mixed letters digits like this:

First 4 digits are converted to 2 letters.
Next 2 digits are kept as is.
Next 4 digits are converted to 2 letters.
Next 2 digits are kept as is.

Examples:

Configuration File TTMHEAD …	**Received Touch Tone digits, BA…**	**Result of combining them**	**Normal representation**
BAxxxxxxxxxxxx	3261 29 7232 78	3261 29 7232 78	EM 29 QE 78
BAxxxxxxxxxx	3261 29 7232	3261 29 7232	EM 29 QE
BAxxxxxx	3261 29	3261 29	EM 29
BAxxxx	3261	3261	EM
BAxx 3261297232	78	3261 29 7232 78	EM 29 QE 78
BAxxxxxx 326129	7232 78	3261 29 7232 78	EM 29 QE 78
BAxxxxxxxx 3261	29 7232 78	3261 29 7232 78	EM 29 QE 78
BAxxxxxx 3261	29 7232	3261 29 7232	EM 29 QE

Even with the greatest precision, of 8 characters, this represents a region roughly 0.28 mile (0.45 km) high (north-south) and up to twice that wide. When converting to Latitude and Longitude we use the center of the region.

5.7 QIKcom-2 Satellite Grid Squares

As described here, http://aprs.org/qikcom-2.html , the first two letters of a Maidenhead locator are represented by two digits. This only includes about 1/3 of the Earth’s surface but it’s the part with nearly all of the human population.

The format must be the “B” button, optionally some other button, and exactly four lower case x characters to match the digits.

```
TTSATSQ    BAxxxx
```

For example, if we received the tone sequence “BA1819” it would be first translated to “FM19” and then to the corresponding latitude and longitude.

6 Comment / Status

This implementation recognizes all standard types:

C*n* - Exactly one digit indicates a predefined status.

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.

You can replace any of the 9 default status messages like with configuration commands like this:

```
TTSTATUS  5  "Clue found"
TTSTATUS  6  "Subject found"
```

When added to the comment of the APRS Object Report, this status is preceded by "/". This makes it easy to distinguish status from any other parts of the comment.

7 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 have been some discussions about a very compact event-specific 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 field of any 5 digits, 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.

Received data:	`78123`
Match to pattern:	`xxyyy`
Definition:	`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. 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 field with exactly 6 digits.

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

Here we are saying that when we receive a field of 6 digits, 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.

Received data:	`781239`
Match to pattern:	`xxyyyz`
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**<u>9</u>***B9**<u>78</u>***AB166*AA2B4C5B3B0A**<u>123</u>**

which is processed as if someone had punched all of that 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**<u>1</u>**23# would match the first one because it requires 1 in the middle position.

The touch tone sequence 78**<u>2</u>**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**<u>3</u>**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.

Macro expansion works on the field level so traditional forms and macros can be combined in a single transmission. 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 audio level = 98(-2/-2)   [NONE]   tt
[0.dtmf] DTMF>APDW13:t9*01123#
Raw Touch Tone Data, --no-symbol--, DireWolf, WB2OSZ
9*01123#
Macro tone sequence: '9'
Matched pattern  13: 'z', x=, y=, z=9
Replace with:        'Cz'
After substitution:  'C9'
Macro tone sequence: '01123'
Matched pattern  10: 'xx1yy', x=01, y=23, z=
Replace with:        'B9xx*AB166*AA2B4C5B3B0A1yy'
After substitution:  'B901*AB166*AA2B4C5B3B0A123'
[0.speech] " Message Received."
[0L] WB2OSZ-13>APDW13,WIDE1-1:;BIKE 123 *210221z4239.68N/07121.87Wb/custom 1!T01!
```

```
DTMF audio level = 97(-2/-2)   [NONE]   tt
[0.dtmf] DTMF>APDW13:tC3*C146520*02223#
Raw Touch Tone Data, --no-symbol--, DireWolf, WB2OSZ
C3*C146520*02223#
Macro tone sequence: '02223'
Matched pattern  11: 'xx2yy', x=02, y=23, z=
Replace with:        'B9xx*AB170*AA3C4C7C3B0A2yy'
After substitution:  'B902*AB170*AA3C4C7C3B0A223'
[0.speech] " Message Received."
[0L] WB2OSZ-13>APDW13,WIDE1-1:;FIRE 223 *210228z4239.62N/07121.87Wf146.520MHz /in service!T02!
```

Let's take the result and feed it into the "decode_aprs" utility.
We see that the symbol is a fire truck and the "!T02!" is recognized as being location "02" from a list of up to 100.

```
$ echo 'WB2OSZ-13>APDW13,WIDE1-1:;FIRE 223 *210228z4239.62N/07121.87Wf146.520MHz /in service!T02!' | decode_aprs

WB2OSZ-13>APDW13,WIDE1-1:;FIRE 223 *210228z4239.62N/07121.87Wf146.520MHz /in service!T02!
Object, "FIRE 223", FIRE TRUCK, DireWolf, WB2OSZ
N 42 39.6200, W 071 21.8700, APRStt location 02 of 100, 146.520 MHz
 /in service
```

```
DTMF audio level = 96(-2/-2)   [NONE]   tt
[0.dtmf] DTMF>APDW13:t03323#
Raw Touch Tone Data, --no-symbol--, DireWolf, WB2OSZ
03323#
Macro tone sequence: '03323'
Matched pattern  12: 'xxyyy', x=03, y=323, z=
Replace with:        'B9xx*AB180*AA3A6C4A0Ayyy'
After substitution:  'B903*AB180*AA3A6C4A0A323'
[0.speech] " Message Received."
[0L] WB2OSZ-13>APDW13,WIDE1-1:;DOG 323  *210236z4239.54N/07121.87Wp!T03!
```

Suppose we wanted to process the **QIKcom-2** format message. It's not necessary to write any new code. Just define a macro like this:

```
TTMACRO   xxxxzzzzzzzzzz   BAxxxx*ACzzzzzzzzzz
```

The field of 14 digits would get broken in to groups of 4 and 10. The first 4 are processed as a satellite grid location. The other 10 are treated as a callsign. Let's look at what happens when this is heard:

```
*18199242771558#
```

```
DTMF audio level = 32(-2/-2)    [NONE]    tt
[0.dtmf] DTMF>APDW12:t*18199242771558#
Raw Touch Tone Data, --no-symbol--, DireWolf, WB2OSZ
*18199242771558#
Macro tone sequence: '18199242771558'
Matched pattern  16: 'xxxxzzzzzzzzzz', x=1819, y=, z=9242771558
Replace with:        'BAxxxx*ACzzzzzzzzzz'
After substitution:  'BA1819*AC9242771558'
[0L] WB2OSZ>APDW12,WIDE1-1:;WB4APR-12*260203z3755.50N\08107.00WA!TBA!
```

The field of 14 digits matches a pattern which splits it into groups of 4 and 10 digits. The first 4 are treated as a satellite square. The other 10 are processed as the new fixed length call format. It's converted to an object and transmitted.

7.1 Text to Tone Sequence Conversions

Constructing macro definitions containing callsign, object names, and symbols can be rather tedious and error prone. For the callsign or object name, it might involve running the "text2tt" utility then copying the result. For the symbols, you need a table from the back of the APRS Protocol Reference. I have one in a binder on my shelf but most people probably don't have it handy. Most people probably don't.

A macro definition can contain these:

- **AC{***callsign***}**

  The specified callsign is converted to the corresponding tone sequence in the fixed length callsign format.

  Example: AC{WB2OSZ}
  Converted to: AC9226711598

- **AA{***objname***}**

  The specified object name is converted to the corresponding tone sequence..

  Example: AA{team 4}
  Converted to: AA8A3B2A6A0A4

  What is the difference between a callsign and an object name? They both end up in the same place, the name in an APRS Object Report packet. A callsign is limited to 6 characters. A default SSID of "-12" is appended to it. An object name can be up to 9 characters and have embedded spaces. No SSID is appended.

- **AB{***symbol***}**

  The symbol descriptions are searched for one that contains the given string. It is replaced by the corresponding code.

Example: AB{canoe}
Converted to: AB135

Example: AB{Jet Ski}
Converted to: AB0835A

How do you know what symbols are available? Run "direwolf –S" (upper case S option) to get a list of the symbol descriptions and their codes.

Let's revisit our earlier example where we have:

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

Instead of handcrafting these:

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

It can be done in a much easier way:

```
TTMACRO  xx1yy  "B9xx*AB{bicycle}*AA{bike }yy"
TTMACRO  xx2yy  "B9xx*AB{fire truck}*AA{fire }yy"
TTMACRO  xxyyy  "B9xx*AB{dog}*AA{dog }yyy"
```

Note that the macro definition contains embedded spaces so it should be enclosed in quotes.

8 Audible Responses

Responses can use either Morse code or speech. The default is Morse code "R" for received correctly or "?" for any sort of error.

Each of the audible responses can configured with a command of the following form:

TTERR *msg_id method text*

Where,

msg_id is message identifier from the table below.

method is either SPEECH or MORSE.

text is what to send over the radio. Should be in quotes if more than one word.

Message Identifier	Meaning
OK	Valid touch tone sequence was processed.
D_MSG	D was first char of field. Messages not implemented yet.
INTERNAL	Internal error. Shouldn't happen.
MACRO_NOMATCH	No definition for digit sequence received.
BAD_CHECKSUM	Bad checksum on call.
INVALID_CALL	Invalid callsign.
INVALID_OBJNAME	Invalid object name.
INVALID_SYMBOL	Invalid symbol specification.
INVALID_LOC	Invalid location.
NO_CALL	No call or object name included in the tone sequence.
INVALID_MHEAD	Invalid Maidenhead Grid Square Locator.
INVALID_SATSQ	Satellite square must be 4 digits.

The default is equivalent to the following configuration file settings.

```
TTERR  OK           MORSE  R
TTERR  D_MSG        MORSE  ?
…
TTERR  INVALID_SATSQ  MORSE  ?
```

To use speech, you must first install a speech synthesizer application and specify an appropriate script file in your configuration file. Examples:

```
SPEECH dwespeak.bat                    -- for Windows
SPEECH dwespeak.sh                     -- for Linux
```

Complete details are in the Dire Wolf User Guide.

If you want voice responses, you could use something like this:

```
TTERR  OK               SPEECH "Message Received O K. "
TTERR  D_MSG            SPEECH "D type not implemented. "
TTERR  INTERNAL         SPEECH "Internal error. "
TTERR  MACRO_NOMATCH    SPEECH "No matching pattern for digit sequence. "
TTERR  BAD_CHECKSUM     SPEECH "Bad checksum on call. "
TTERR  INVALID_CALL     SPEECH "Invalid call sign. "
TTERR  INVALID_OBJNAME  SPEECH "Invalid object name. "
TTERR  INVALID_SYMBOL   SPEECH "Invalid symbol. "
TTERR  INVALID_LOC      SPEECH "Invalid location. "
TTERR  NO_CALL          SPEECH "No call or object name. "
TTERR  MHEAD            SPEECH "Invalid Maidenhead Locator. "
TTERR  SATSQ            SPEECH "Satellite square must be 4 digits. "
```

8.1 Invoking your own application.

You might want to perform other processing besides generating an APRS Object Report packet.

You might want to generate an audible response based on the particular call, location, etc.

Both of these can be accomplished by specifying a command to be run when a valid tone sequence is received. The configuration file option is TTCMD followed by the command to run. It can be a simple command (appropriate for your operating system) or the name of something more interesting written in bash, python, perl, or your other favorite language.

The command can use the values of these environment variables to change its behavior. If it generates any text output, that is used as the response to the user.

Environment Variable	Meaning	Examples
TTCALL	Callsign or object name.	WB2OSZ TEAM A
TTCALLSP	Callsign or object name split into separate characters so speech synthesizer would not try to pronounce it as a word.	W B 2 O S Z A P R S (You wouldn't want it coming out sounding like APE PURRS!)
TTCALLPH	Callsign or object name.	whiskey bravo two oscar sierra zulu
TTSSID	SSID to distinguish multiple objects with same callsign.	12
TTCOUNT	Number of times this call / object name heard. 1, for the first time, could be used to send a greeting.	1
TTSYMBOL	Symbol table/overlay and symbol.	/' for airplane

		Js for Jet Ski
TTLAT	Latitude.	39.202083
TTLON	Longitude.	-94.604167
TTFREQ	Frequency from the **C***nnnnnn* tone sequence.	146.520MHz
TTCOMMENT	Comment.	TEMPUS FUGIT
TTLOC	Original location if other that latitude and longitude. This is useful because the original might have specified a large region.	EM29QE78 32TPP81794936 19T 305440 4725830
TTSTATUS	Status from the **C***n* tone sequence.	Off duty
TTDAO	Enhanced location description so you can distinguish between predefined points, the corral, etc.	!T12!

Configuration file example for Windows:

```
TTCMD  c:\strawberry\perl\bin\perl.exe  sar.pl
```

Configuration file example for Linux:

```
TTCMD  sar.pl
```

This is the “sar.pl” file mentioned:

```
#!/usr/bin/perl

# Simple script to generate custom speech responses for APRStt.
# First, gather information of interest.

$name = $ENV{TTCALL};
$loc = $ENV{TTLOC};
$status = $ENV{TTSTATUS};

# Change pronounciation of team names.

if ($name eq "TEAM A") { $name = "Team Alpha"; }
if ($name eq "TEAM B") { $name = "Team Bravo"; }
if ($name eq "TEAM C") { $name = "Team Charlie"; }

# Extract the desired digits from the location.

($zone,$east,$north) = split " ", $loc;
$east = substr("000000" . $east, -4, 3);
$east =~ s// /g;
$north = substr("000000" . $north, -4, 3);
$north =~ s// /g;

# Combine into string to be spoken.

print "$name, $east east, $north north. $status\n";
```

9 Object Report Generation and Routing

The tones received over the radio and configuration settings are combined to generate an APRS "Object Report" packet looking something like this:

```
WB2OSZ-13>APDW13:;TEAM C   *151247z4239.62N/07122.43We[19T 305440 4725830]
/Clue found !TB6!
```

- The source is the call of the gateway station.
- Destination is the software version.
- Data Type Indicator of ";" means object report.
- The object name is the APRStt user's call or object name provided.
- Time stamp.
- Latitude, Longitude, and symbol.
- Comment. In this case it has 3 components:
    - Original location specified.
    - Status from the "Cn" tone sequence.
    - Enhanced location information. We see the location tone sequence was "B6…"

The conversion is enabled by a command of this form:

`TTOBJ` *rec-chan send-to [via-path]*

Where,

rec-chan is the channel where the DTMF decoder is listening.

send-to indicates where the result should be sent. It can be one or more of these separated by commas.

- Transmit radio channel.
- APP - meaning send to any attached application(s).
- IG - meaning send to APRS-IS server if IGate is configured.

via-path is the digipeater path used when transmitting. Must NOT contain any spaces.

Examples:

```
TTOBJ  1  0  WIDE2-1,WIDE1-1
TTOBJ  1  0
TTOBJ  1  APP,IG
```

In all of these examples, the DTMF decoder is listening on channel 1. In the first two cases APRS packets will be transmitted on channel 0. It is possible, but unusual, for both channels to be the same. You can optionally add a digipeater path. In the last case, the object reports are sent to any attached applications and an APRS-IS server if the IGate feature is enabled but not transmitted over the radio.

APRS is inherently unreliable. We throw packets out there and, in most cases, don't know if they were heard by anyone. For this reason, the packets are transmitted multiple times with increasing intervals in between. The first transmission is delayed by 3 seconds. The object report is sent again after delays of 16 seconds, 32 seconds, 64 seconds, 2 minutes, and 4 minutes. The time stamp remains constant because it indicates when the information was new not when it is being passed along.

Sending information to an attached application or APRS-IS server is more reliable. This is done immediately and only once.

9.1 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, preferably 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.

9.2 Enhanced position reporting

APRS has many different ways of specifying a location:

- One of many predefined locations.
- x & y coordinates in some grid.
- Bearing and distance from a known point.
- Different size grid squares covering large areas.
- No known location.

All of these get converted to a specific latitude and longitude in the object report. The behavior is a good approximation in some situations and backward compatible with existing systems, but it has some

disadvantages. In some cases, the conversion loses information. In some cases, the conversion supplies a made up location. How can the receiving station tell the difference?

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, such as a table rather than on a map, 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 where multiple objects might be at the same location. We might 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 location. 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 location is unknown. (the "corral" lat/lon were assigned by the gateway.)
`!T`*n*` !`	The location is one of ten predefined locations (i.e. the B0n tone sequence) (where 'n' represents some digit 0-9.)
`!T`*nn*`!`	The location is one of 100 predefined locations (i.e. the B9nn tone sequence)
`!TB`*n*`!`	The location was specified by some other method. n is the character following B in the location. For example, “!TB5!” indicates the location came from a bearing and distance.

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.

9.3 Location Ambiguity

“Object Reports” always have a very specific Latitude/Longitude location. The original DTMF sequence might represent a large area.

If using a Maidenhead Locator, we are specifying some rectangular area. The generated latitude and longitude represent the center point. How large is the region? The original intent can be preserved by putting the Maidenhead Locator in the “object report” packet somehow. Only the MIC-E and “Status Report” formats have a standard way of representing this. Random text, in the comment, would be instantly recognizable by a human, but without a standard representation, display applications probably would not be able to take advantage of that extra information. I’ve seen references to putting it in square brackets thusly, [cm87xi], so I will follow that convention until something better comes along.

When USNG/MGRS coordinates with few digits, we are also specifying a rectangular area. How big is it?

In both cases, the original location will be placed in the comment field surrounded by [and] characters.