The ISCAT Encoding Process

ISCAT was introduced into the WSJT Suite from Version 9.3 (the latest version at the time of writing). The mode is intended for contacts at V/UHF and higher and is aimed at the paths where short duration openings or brief periods of weak signal path exist, such as Troposcatter, Aircraft Reflection and Meteor pings. It is designed to be able to decode on short bursts, but also take advantage of averaging over longer periods of weaker reception.

In concept it has more in common with FSK441 than the other WSJT modes such as JT4 and JT6 in that there is no Forward Error Correction (FEC), although there is a degree of soft post-processing through averaging that FSK441 does not offer. Messages can be of arbitrary length up to 28 characters. A header and framing sequence is inserted at regular intervals and the message interspersed between these.

Transmission timing is essentially irrelevant, but the decoder in the WSJT software works with 30 second, or optionally 15 second, periods, so for beaconing and transmission purposes it may be convenient to keep with similarly timed bursts rather than just sending continuously.

Two versions of ISCAT are in use, ISCAT-A and ISCAT-B, varying only in tone spacing and symbol rate.

The transmission consists of 1 of 42 sequential tones, with each tone corresponding to a character taken from the restricted alphabet below.

0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ /.?@-
[space] lies immediately after Z at no 36

For the –A version, tone spacing is 11025 / 512 = 21.533Hz and the symbol rate is the reciprocal of this, or 46.44ms. The –B version uses double the tone spacing, 43.066Hz with 23.2ms symbols. The lowest, Tone 0, is placed at 1012Hz and 559Hz respectively, giving resulting frequency spans of 1012 - 1938Hz and 559 - 2411Hz (Exact Tone0 values are 47 and 13 times their respective spacing)

Generating and Formatting the message.

"ISCAT messages are repeated as many times as will fit into a Tx sequence. The sync pattern and a message-length indicator have a fixed repetition period. The user message starts with a start-of-message symbol, '@', and continues with the character-by-character message (no FEC). This part repeats at its own natural length."

The ISCAT encoding process is a bit confusing as it involves two non-related repeat sequences. First of all there is a repeat period every 24 symbols; this is fixed whatever the message length and is based on a block with a six-symbol header and 18 symbols of message data.

The header is made up of a Costas Array consisting of the four characters, or values, 0,1,3,2 followed by two more characters. These 5th and 6th elements are the length of the message (including the BOM character, see below) and the length plus 5.

The second repeat interval is derived from the message length. The message is first preceded by a Beginning Of Message (BOM) symbol, (the @ character / tone 40). The repeated BOM + message is broken up and inserted into the 18 symbol periods after the header. So unless the total message length, including the BOM symbol has 6 as a factor, a complicated repeat sequence will result. The decoder copes with this!

The best way to describe the encoding process is by two examples:

To encode a message into the correct sequence of tones, first convert each character to a Tone value from 0 to 41 (Illegal characters are replaced by spaces). Add a BOM (@, value 40) to the start:

If the message is *G4JNT IO90IV TEST* add a @ for the BOM, so the message becomes *@G4JNT IO90IV TEST* This message length is 18 characters; chosen very deliberately! The resulting tone numbers are *40, 16, 4, 19, 23 … etc...etc…28, 29.*

The modified message is repeated by stacking end to end for the duration of the complete Tx period (indefinitely if you want). The marker, and the length which is sent in a more robust way, are used in the decoder to detect beginning and end of each message repeat, allowing overlay and averaging for improved reliability.

Generate the six character header block which is *0,1,3,2,18,23* (Costas array, len, len + 5)
Or expressed in terms of symbols, *0123IN*

To transmit the message, start with this block of six then send the next 18 of the message, then the 6 symbol head, next message 18 and so on. Now see why that message length was chosen! Choosing a 17 character message (total 18 symbols) means every 24 symbols are absolutely identical and will give a very characteristic repeating sound. The total tone pattern sent over the air interface is :

0 1,3, 2,18,23,40,16,4,19 23…etc...etc…28,29,0,1,3,2,18,23,40,16,4,19,23…etc...etc…28,29…
Repeating indefinitely.

Or in terms of the equivalent characters :

0123IN@G4JNT IO90IV TEST0123IN@G4JNT IO90IV TEST0123IN@G4JNT IO90IV TEST0123IN@G4JNT IO90IV TEST0123IN@G4JNT IO90IV TEST0132 ……..

Now consider a different length message, with its BOM marker:

@G8IMR TESTING- ISCAT-A which has length 23.

The header is now 0,1,3,2,23,28 or if expressed as equivalent characters, *0132NS*
The header must repeat every 24 symbols, with the remaining 18 taken sequentially from the concatenated message. So now a totally different structure results. (For clarity, this is illustrated only in the character equivalent terms).

0132NS@G8IMR TESTING- IS0132NSCAT-A@G8IMR TESTIN0132NSG- ISCAT-A@G8IMR T 0132NS ESTING- ISCAT-A@G80132NS IMR TESTING- ISCAT0132NS -A@G8IMR TESTING-

Listening by ear the sound still has a noticeable rhythmic element from the repeated header, approximately every 1.1 seconds (0.56s for the –B mode), but the overall sound is more unstructured.

Encoding Algorithm

Set up a repeat loop , repeating every symbol – in real time this could be a timer interrupt at the symbol rate. Off line, it would be a DO or FOR loop for the required number of transmitted symbols.

```
BlockCount   =  0                At the start of the Tx period, initialise these two counters
MessageCount  =  0
Loop            indefinitely, or the required number of symbols

        BlockCount counts Modulo 24 repeatedly,   or 0 to 23
        MessageCount   counts repeatedly   0 to (Total message length –1)

        If  BlockCount  MOD  24  <  6   generate the appropriate header symbol
        Get the Header Character,  0 – 3 = Costas data, 4 =  length, 5  = Length+5
ELSE                         for counts 6 to 23
        Char = (Message String ,  MessageCount , 1)      get the  next message character
        (MessageCount  = MessageCount + 1 ) MOD Length   'Increment message count MODULO Length
   end if
end loop
```

Decoding

The decoder first of all sorts out the 24 symbol repeat period based on finding the Costas array symbols, and extracts the length. It then looks for the BOM marker and tryst to extract a message. To quote the designer :

"The decoder defines a block size as short as a single un-repeated message, and tries to decode. It steps through the entire transmission this way; then does it again using 2x the block length, then 4x, etc. For each of these it averages over all available repetitions of the message. The "best" decode, according so an heuristic criterion, is finally displayed for the user."

PIC Implementation

The simple encoding process means that instead of storing pre-computed symbols as is done for JT4, 65 and WSPR beacons, the tone numbers can easily calculated on the fly from a message stored as raw text characters, then inserted into a timed stream with the header data as shown in the algorithm above. The generated tone number is then used to calculate frequency data to send to the synthesizer.

Being able to build and change messages on the fly means the mode is suitable for transmission of changeable data from beacons, such as telemetry or warning messages.