GMDSS DSC Messages

A look at the composition of DSC messages and the analysis of received signals

Part One : DSC Messages

The general picture

DSC (Digital Selective Calling) is a method for ships and Coast Stations to initiate calls for routine traffic messages, to give position reports, to initiate telephone connections etc. but mainly for Distress Alerts. The signals are transmitted on a variety of frequencies in the MF/HF and VHF bands. This guide focusses mainly on MF/HF DSC. The signals are sent using Frequency Shift Keying (170Hz shift/100 baud) and the centre frequencies used for "Safety" signalling are listed below.

- 2187.5 kHz
- 4207.5 kHz
- 6312.0 kHz
- 8414.5 kHz
- 12577.0 kHz
- 16804.5 kHz

There are many other frequencies where DSC signals may be found, for example 2177.0 kHz – but these frequencies are less heavily used, and are for routine calling, rather than for Distress and Urgency. The majority of Coast Stations around the world do not monitor the secondary DSC channels, and as a result most activity is to be found on the standard GMDSS channels.

There is a requirement under GMDSS for all vessels to do a live over-the-air test of their DSC systems, on a weekly basis, and preferably by a test call with a Coast Station. As a result the majority of signals heard on the air are test calls, and their resulting acknowledgements. This at least gives ample sources of signals for us to monitor.

Stations (Ship, Coast etc.) identify themselves in DSC by use of their allocated MMSI number.

The MMSI – Maritime Mobile Service Identity

This is a 9-figure numerical code, issued to ships, Coast Stations and various Aids to Navigation etc. The MMSI uniquely identifies the station, and also identifies the Country of registration, as well as the type of station. The country is identified by a three digit code – the [MID] (Maritime Identification Digits).

The MMSI is made up as follows

- Coast Stations : **00MIDXXXX** – two leading zeros, three digits of the "MID" and four digits to make up a unique 9-digit MMSI
- Ship Stations: **MIDXXXXXX** – Three digit "MID" followed by six digits to make up the unique 9-digit MMSI
- Groups of Ships: **0MIDXXXXX** – One leading zero, three digit "MID" and 5 figures to make up the unique 9-digit MMSI.
- Aids to Navigation : **99MIDXXXX**
- Craft associated with a parent ship: **98MIDXXXX**
- Aircraft using MMSI for Search & Rescue: **111MIDXXX** (fixed wing), or **111MID5XX** (helicopters)

The "MID" identifies the country, and a selection of examples is below

- [232] [233] [234] [235] : United Kingdom
- [219] [220] : Denmark
- [338] [366] : USA

Example MMSIs

002320017 : This is Coast Station (two leading zeros), from the UK (MID = 232). The MMSI belongs to Milford Haven Coastguard.

636014168 : This is a Ship (no leading zeros/99/111). Liberian registered (MID = 636). The MMSI belongs to the "CMA CGM Opal" a Liberian Container Ship.

It is because of the structure of MMSIs that software such as YaDD and DSCDecoder are able to indicate whether a MMSI received belongs to a Coast Station or a vessel, and the country of origin of the MMSI – even without necessarily knowing anymore about the station.

DSC messages

The information transmitted in a DSC message includes

- The Sender's MMSI number
- The addressee – All Ships, Stations within a specific Geographical Area or an individual station's MMSI
- The Format of the message – Geographical area, Distress, All Ships, Individual Call etc.
- The Category of the message – Routine, Safety, Urgency or Distress
- "Telecommands" – additional information for the recipient
- Data – a frequency/channel for communications, current position, nature of Distress etc.
- End of Sequence – does the message require an acknowledgement from the addressee, is the message an Acknowledgement or is no further action required from receiving stations?
- A checksum for determining if the message has been received without errors.

The general format of a DSC Message

Dotting Pattern	DX/RX Phasing Sequence	A Format Specifier 2 identical Characters	B Called Party Address 5 Characters	C Category 1 Character	D Self-Identity 5 Characters	E TC1 TC2 2 Characters	F Freq Info 3 Characters	G Freq Info 3 Characters	H End of Sequence 3 Identical DX plus 1 RX Character	I Error Check 1 Character

Symbols

Each piece of a DSC message is allocated a three-digit "Symbol" value. This is a number between 000 and 127. These "DSC Symbols" are the heart of the protocol, and are the means of conveying many different types of message. The different parts of the message (Addresses, Message Format, Category, Data etc.) are coded into specific symbol values – always between 000 and 127. We'll look at how each part of a message is coded into symbols next.

Addresses – MMSI numbers

The 9-digit MMSI number is converted to **FIVE DSC Symbols** as follows

MMSI **002320017**

Split into five segments **00 23 20 01 7**

Add "**padding zeros**" to form five 3-digit symbols **000 023 020 001 070**

Another example

MMSI **636014168**

Split into five segments **63 60 14 16 8**

Add "**padding zeros**" to form five 3-digit symbols **063 060 014 016 080**

We'll see how the MMSI, in DSC symbol form, is incorporated into a message later. The important thing here is to see that everything in a DSC message is carried in 3-digit symbols, with a value between 0 and 127.

Now we'll look at the component parts of a DSC message.

Format

The Format defines whether a message is a "Selective Call to an individual station", an "All ships call", a "Geographical area call" etc. Each "Format" is given a specific symbol value.

Format Values

Format	Meaning
102	Geographical Area
112	Distress
114	Ships having common Interest
116	All Ships
120	Selective Call to an Individual Station
123	Individual Station semi-automatic/automatic

The receiver looks at the symbol which carries the "Format" to determine what type of message is being sent, and then knows how to interpret the following symbols correctly.

Category

Messages can have one of four "Category" values which show the importance of the message. Each different Category is allocated a specific number.

Category Values

Category	Meaning
100	Routine
108	Safety
110	Urgency
112	Distress

Telecommands

Messages also contain "Telecommand" values – there are two telecommand words in a message "**Telecommand One**" and "**Telecommand Two**" giving a wide scope for signalling to the receiver what the sender would like to happen next. Many are redundant, or rarely ever used, but the full lists are below.

Telecommand One (TC1) Values

Telecommand One	Meaning
100	F3E/G3E All Modes TP
101	F3E/G3E duplex TP
102	Polling
104	Unable to comply
105	End of Call
106	Data
109	J3E TP
110	Distress Acknowledgement
112	Distress Relay
113	F1B/J2B TTY-FEC
115	F1B/J2B TTY-ARQ
118	Test
121	Ship Position update
126	No information

These Telecommands will inform the receiver that for instance, with TC1 = **109**, that the sender wishes to continue subsequent communications in J3E Telephony, i.e. SSB voice.

Some DSC messages will use the Second Telecommand in addition, although that's not often seen

Telecommand Two (TC2) Values

Telecommand Two	Meaning
100	No reason Given
101	Congestion at maritime switching centre
102	Busy
103	Queue Indication
104	Station barred
105	No operator available
106	Operator temporarily unavailable
107	Equipment disabled
108	Unable to use proposed channel
109	Unable to use proposed mode
110	Ships and aircraft of states not parties to an armed conflict
111	Medical Transports
112	Pay-phone/public call office
113	Facsimile/data
126	No information

Telecommand Two is generally associated with using DSC to initiate ship to shore "public correspondence" calls, rather than in its use as a safety signalling system, and these TC2 symbols are rarely seen.

The End of Sequence symbol

This is an important part of a message, and tells the recipient whether the sender wants a response or not.

There are three possible values for the End Of Sequence (**EOS**) symbol.

If a message requires the receiver to send an "acknowledgement of receipt" then the transmitted EOS symbol is "RQ" with a value of **117**. This is displayed in YaDD and DSCDecoder as "**REQ**". The sender **REQ**uires an acknowledgement.

If a message is sent in reply to such a "**REQ**" message it will have an EOS of "BQ", with a value of **122**. This is displayed in YaDD and DSCDecoder as "**ACK**". The message is sent in **ACK**nowledgement.

The vast majority of MF/HF DSC messages are TEST calls, and the initial TEST message will have an EOS of **117** – a REQ. The replying message, usually from a Coast Station, containing the TEST acknowledgement, will have an EOS of **122** – an **ACK.**

Some messages are sent without the need for anyone to reply in acknowledgement. Messages such as "All Ships" or "Geographical Area Call" messages – perhaps advertising an impending Gale Warning announcement – will not require any further acknowledgements. These messages have an EOS of **127.** This is shown simply as "**EOS**".

End Of Sequence Values

End of Sequence	Meaning
117	Ack RQ (REQ)
122	Ack BQ (ACK)
127	EOS

Building a message – a worked example.

The general format of a DSC Message

Dotting Pattern	DX/RX Phasing Sequence	A Format Specifier 2 identical Characters	B Called Party Address 5 Characters	C Category 1 Character	D Self-Identity 5 Characters	E TC1 TC2 2 Characters	F Freq Info 3 Characters	G Freq Info 3 Characters	H End of Sequence 3 Identical DX plus 1 RX Character	I Error Check 1 Character

An "Individual, Safety, Test" Message from a vessel

We'll use the chart above to build a message, taking the required symbol values from the previous tables.

The scenario is that a vessel wants to send a Test Message to a Coast Station. He wants the recipient to respond with acknowledgement, but doesn't have any requirement for further communications, and has no position information to pass on.

A Format : *Individual Stations Call* **120**

C Category : *Safety* **108**

E Telecommand One: *Test* **118**

E Telecommand Two : *No info* **126**

H EOS : *REQ* (*RQ*) **117**

D Sender's MMSI : **235448000** (M.V Hrossey – callsign VSTY6)

B Destination MMSI : **002320001** (Shetland Coastguard)

We can now slot these values into the correct places to build the message.

The 9-digit MMSIs are split across 5 DSC symbols with "padding zeros" to form 3-figure symbols

The "To MMSI" of 002320001 encodes as: 000 023 020 000 010

The "From MMSI" of 235448000 encodes as: 023 054 048 000 000

The basic message is now:

A	B	B	B	B	B	C	D	D	D	D	D	E	E
120	**000**	**023**	**020**	**000**	**010**	**108**	**023**	**054**	**048**	**000**	**000**	**118**	**126**

We then add 6 extra symbols, which could in other circumstances be used to carry lat/long or frequency information. In this message we don't need to convey any information, so we use "126" (No Info).

We also add the EOS of **117**

120 000 023 020 000 010 108 023 054 048 000 000 118 126 126 126 126 126 126 126 117

This is the basic message, using twenty-one DSC symbols.

There's one important symbol missing.

The Error Check Character

To allow the receiver to have confidence that the message had arrived without errors the transmitting station adds an Error Check Character (**ECC**), which is a calculated value using the numerical values of the symbols in the rest of the message. The receiver can than perform the same calculation, using the symbols it receives, and compare the result with the ECC received in the message. If they agree then there's a strong probability that no errors have occurred, and that the message contents are valid.

The ECC is calculated using the XOR logical operator.

A DSC message contains various symbols, each with a specific meaning, and a numerical value between 0 and 127. The symbols therefore can be represented as 7-bit binary numbers.

Decimal **0** = Binary **0000000**

Decimal **127** = Binary **1111111**

Binary numbers can be manipulated with logical operators (AND, NOR, OR, XOR etc.) and the ECC calculation in DSC is done using the XOR (Exclusive OR) operator.

The truth table for the XOR operator

A	B	XOR
0	0	0
1	0	1
0	1	1
1	1	0

The ECC is calculated by finding the result of successively ***XORing each symbol in turn***.

For a simple example suppose we want to find the result of "**102 xor 99**"

Convert 102 to binary **1100110**

Convert 99 to binary **1100011**

Look at each bit-position in turn and use the XOR truth table to decide on the XOR value

1 1 0 0 1 1 0

1 1 0 0 0 1 1

= = = = = = =

0 0 0 0 1 0 1

The XOR result is binary **0000101** which is decimal **5**

How XOR detects errors

In our example above, we have two data symbols (**102** and **99**) and an ECC symbol (**5**).

If one of those symbols is decoded with an error, how does the XOR function detect it?

The "message" is transmitted as **102 099 005** and received as **103 099 005**.

Convert 103 to binary **1100111**

Convert 99 to binary **1100011**

Look at each bit-position in turn and use the XOR truth table to decide on the XOR value

1 1 0 0 1 1 1

1 1 0 0 0 1 1

= = = = = = =

0 0 0 0 1 0 0

The XOR result is binary **0000100** which is decimal **4**

The cECC (calculated ECC) of **4** no longer matches the received ECC of **5**.There's an error, somewhere.

Suppose there are TWO errors

Imagine that we received **101 099 004**

Convert 101 to binary **1100101**

Convert 99 to binary **1100011**

Look at each bit-position in turn and use the XOR truth table to decide on the XOR value

1 1 0 0 1 0 1

1 1 0 0 0 1 1

= = = = = = =

0 0 0 0 1 1 0

The XOR result is binary **0000110** which is decimal **6**

Our message 101 099 004 must be corrupt – the cECC is **6** but the received ECC is **4.** The XOR has detected errors – even when there are more than one, and when one is in the ECC symbol.

Fly in the ointment - aside

Imagine that we received **103 098 005**

There are two errors compared to our genuine **102 099 005**

Convert 103 to binary **1100111**

Convert 98 to binary **1100010**

Look at each bit-position in turn and use the XOR truth table to decide on the XOR value

1 1 0 0 1 1 1

1 1 0 0 0 1 0

= = = = = = =

0 0 0 0 1 0 1

The XOR result is binary **0000101** which is decimal **5**

The cECC matches the Received ECC – but there are two errors! DSC is not infallible, but this situation is very unlikely **in a real message**, where there are 20 or more symbols. It is very unlikely that a specific combination of errors will still yield a correct ECC comparison, with such a large number of symbols involved in the calculation.

In general, for the ECC to match correctly there must be no “symbols in error”.

For a real message we need to do the XOR operation on each symbol in turn, one after the other, until we've XORed all the symbols, and we have an overall value for the ECC.

Back to the fray

We are creating a DSC message for transmission, so calculate the true ECC value for the message.

In our DSC Message we could do the calculation symbol by symbol as above, convert each symbol to binary, carry out the XOR on each bit-position, use the result to XOR with the next DSC symbol expressed in binary…. until we've dealt with all the symbols in the message, then convert back to decimal.

Method 1

The long-handed way to calculate this is by writing the symbols in binary and then counting the number of "ones" in each column – an odd number of ones gives a result of "1" and even number of ones gives a result of "0" ("all zeros" count as "even")

```
120 = 1 1 1 1 0 0 0

000 = 0 0 0 0 0 0 0

023 = 0 0 1 0 1 1 1

020 = 0 0 1 0 1 0 0

000 = 0 0 0 0 0 0 0

010 = 0 0 0 1 0 1 0

108 = 1 1 0 1 1 0 0

023 = 0 0 1 0 1 1 1

054 = 0 1 1 0 1 1 0

048 = 0 1 1 0 0 0 0

000 = 0 0 0 0 0 0 0

000 = 0 0 0 0 0 0 0

118 = 1 1 1 0 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0
```

```
126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

117 = 1 1 1 0 1 0 1
```

ECC = 1 1 1 0 0 0 1 = decimal 113

Method 2

The quick method for calculating the ECC is to use a calculator that understands binary and logical operators. Not all calculators can do it, but some can, and some calculator Apps for iPhone/iPad and Android can too. I use the free Android app **"Mobi Calculator"**. An iPad app that supports XOR calculation is **"TouchCalc".**

120 XOR 000 = *120*

120 XOR 023 = *111*

111 XOR 020 = *123*

123 XOR 000 = *123*

123 XOR 010 = *113*

113 XOR 108 = *029*

029 XOR **023** = *010*

010 XOR **054** = *060*

060 XOR **048** = *012*

012 XOR **000** = *012*

012 XOR **000** = *012*

012 XOR **118** = *122*

122 XOR **126** = *004*

004 XOR 126 = *122*

122 XOR 126 = *004*

004 XOR 126 = *122*

122 XOR 126 = *004*

004 XOR 126 = *122*

122 XOR 126 = *004*

004 XOR 117 = **113**

The ECC for our message is **113** (in decimal)

The message now gets this ECC value (**113**) added to the end.

The basic "RAW" message

120 **000 023 020 000 010** **108** **023 054 048 000 000** **118 126** 126 126 126 126 126 126 **117** **113**

The Format and the EOS are important symbols to the overall meaning of a message, so they are repeated, the Format symbol is repeated at the beginning, and the EOS symbol is repeated twice at the end.

120 120 **000 023 020 000 010** **108** **023 054 048 000 000** **118 126** 126 126 126 126 126 126 **117** **113** **117 117**

This is the message that is transmitted over the air – after a few more error prevention techniques are brought into play. *Glossing over these, for now*….

The receiver, when presented with the sequence of message symbols

120 **000 023 020 000 010** **108** **023 054 048 000 000** **118 126** 126 126 126 126 126 126 **117**

will use them to calculate its own version of the ECC, using the same technique (XORing each symbol in turn) and will compare the result with the ECC symbol taken from the received message. If they agree then we are confident our received message matches the one that was transmitted. If they don't agree then there's been an error in decoding one or more symbols – and that of course includes the ECC symbol itself.

Let's continue building our message, ready for transmission.

More error detection / prevention

Symbols and Parity and the "10 to 7-bit Parity Test"

We know that the DSC message is composed of "symbols", numbers between 0 and 127 which carry the information and that any symbol, with a value between 0 and 127, can be represented in binary with seven bits:

Decimal **0** = Binary **0000000**

Decimal **127** = Binary **1111111**

Each DSC symbol is therefore a 7-bit number. To allow for detection of bit errors an extra three bits, "the parity bits", are added to each 7-bit symbol, prior to transmission. This allows the receiver to perform a "parity check" to determine that the symbol has (probably) been received correctly.

The parity check is a number between 0 and 7, expressed in 3-bit binary, and is a count of the number of "zeros" in the original binary 7-bit DSC symbol.

The Parity Check allows us to determine, to a degree, whether an individual symbol has been received correctly, even before we've got the whole message, and before we can carry out the overall ECC check.

Let's look at how we convert a **7-bit DSC Symbol** into a **10-bit word with parity**.

Using the message that we're building:

120 120 000 023 020 000 010 108 023 054 048 000 000 118 126 126 126 126 126 126 126 **117 113 117 117**

The first symbol **120** in 7-bit binary is:

120 = 1111000

We count the number of "zeros" in the 7-bit symbol. There are THREE.

The parity check bits are therefore the binary for 3 = **011**

To make the actual 10-bit word that we want to transmit we reverse the order of the original 7-bits and then add the new 3-bit parity bits to the end:

Our 10-bit parity protected word is **0001111011**

How does the parity check help us?

Passing a Parity Check

We'll "reverse engineer" the 10-bit word back to our original DSC Symbol.

0001111011

The last 3 bits represents the number of zeros we expect to find in the first 7-bits (the real data).

011 in binary = **3** in decimal.

We expect 3 zeros in the remainder of the 7-bits : **0001111** and there are indeed 3 zeros. Our word has "passed the parity test". We reverse the order **1111000** and convert to decimal = **120.**

We now know how to check a 10-bit word for "parity errors" and to re-create the original DSC Symbol, if the parity check is "good".

Failing a Parity Check

Suppose we received the 10 bit word **0001011011**

Can we check if it's a valid word?

The parity bits **011** tell us to expect THREE zeros in the main part of the symbol. There are actually FOUR zeros. The word is corrupt and must be ignored.

Suppose we received **0001111010**

Can we check if this one is valid?

The parity bits **010** tell us to expect TWO zeros in the main part of the symbol. There are actually THREE. The word is corrupt, and also must be ignored.

Passing a Parity Check, even when there is an error

Suppose we received this 10-bit word **1001111010**

The parity bits **010** tell us to expect TWO zeros in the main part of the symbol. There ***are*** actually TWO zeros. The word has passed the parity check.

Suppose though that this was originally transmitted as **0001111011 (120)**

Comparing the two copies:

1001111010

0001111011

Two bits are different. The received word **1001111010**, when converted back to 7-bits, and reversed, becomes **1111001**, which is decimal **121**.

This is the incorrect value – although it's "passed the parity test". Two bits being swapped can lead to false positives. This is where the overall ECC comes to the rescue. The false value of 121 for one of the symbols would lead to the overall ECC check failing, and the knowledge that the message contained errors.

Time Diversity Interleaving : The DX and RX copies

Each 10-bit word is sent twice, to give the receiver two opportunities to get an accurate version of the DSC symbol. The symbols are sent once in what is called the "**DX**" position, and after four other symbols have been transmitted they are sent again, in the "**RX**" position. The bit-rate of MF/HF DSC is such that the intervening four symbols (of 10 bits each) between the **DX** and **RX** copies of any symbol take 400ms – so each symbol is sent twice spread out by 400ms. A burst of noise, or a fade, of less than this duration won't wipe out both the **DX** and **RX** copies of any symbol. Even if one copy is missing, ***as long as the other copy is intact*** we can still reconstruct the message. The **Parity Check** is thus a valuable tool for deciding whether to discard one or other of the **DX** or **RX** copies.

To illustrate the application of the diversity interleave, we can inspect a real message, received off-air. The symbols available, before the software *de-interleaves* the **DX** and **RX** copies are as follows

```
125 107 125 106 120 105 120 104 037 120 005 120 005 037 ~~~ ~~~ ~~~ ~~~ ~~~
000 037 000 ~~~ 108 ~~~ 037 000 005 000 005 118 000 ~~~ ~~~ ~~~ 118 126 126
126 126 126 126 126 126 126 126 122 126 102 126 122 122
```

Splitting it up to show the **DX** and **RX** positions, and numbering each symbol

dx	rx	dx	rx	dx1	rx	dx2	rx	dx3	rx1	dx4	rx2	dx5
125	107	125	106	120	105	120	104	037	120	005	120	005
rx3	dx6	rx4	dx7	rx5	dx8	rx6	dx9	rx7	dx10	rx8	dx11	rx9
037	~~~	~~~	~~~	~~~	~~~	000	037	000	~~~	108	~~~	037
dx12	rx10	dx13	rx11	dx14	rx12	dx15	rx13	dx16	rx14	dx17	rx15	
000	005	000	005	118	000	~~~	~~~	~~~	118	126	126	
dx18	rx16	dx19	rx17	dx20	rx18	dx21	rx19	dx22	rx20	dx23	rx21	
126	126	126	126	126	126	126	126	122	126	102	126	
dx24	rx22											
122	122											

*The initial few symbols (**125 107 125 106** 120 **105** 120 **104**) are part of the "phasing" section, and are used by the receiver to find the start of the message, and the boundaries between each 10-bit word.*

Symbols shown as ~~~ are ones that failed the "**parity test**" and have therefore been "lost". There are TEN such missing symbols, with FOUR of these occurring in succession.

Surely the message is corrupt and useless?

If we look for each message symbol in turn, and see where its DX and RX copies are, we find that the message is actually intact!

```
 1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19
120 120 037 005 005 000 000 108 037 005 005 000 000 118 126 126 126 126 126

 20  21  22  23  24
126 126 122 102 122
```

Symbols in RED were received in the "DX" position (the first copy) and those in BLUE were "recovered" from the "RX" position (the second copy). Overall no symbols were missed, despite what seemed like a lot of missing data.

The message decoded correctly

```
19:30:36> 2187.5: 120 120 037 005 005 000 000 108 037 005 005 000 000 118
126 126 126 126 126 126 126 122 102 122
19:30:36>           FORMAT: SELECTIVE CALL
19:30:36>              CAT: SAFETY
19:30:36>               TO: SHIP  370505000
19:30:36>             FROM: SHIP  370505000
19:30:36>              TC1: TEST
19:30:36>              TC2: NO INFO
19:30:36>             FREQ: --
19:30:36>              POS: --
19:30:36>              EOS: ACK
19:30:36>             cECC: 102 OK
```

The Message – almost ready for transmission

To build the message we've taken the following steps

- Create the basic message : **120 000 023 020 000 010 108 023 054 048 000 000 118 126** 126 126 126 126 126 126 **117**
- Calculate the ECC : **113**
- Add copies of the Format and EOS characters
- Interleave the DX and RX copies – separated by 4 intervening words

This gets us here:

```
120 xxx 120 xxx 000 120 023 120 020 000 000 023 010 020 108 000 023
010 054 108 048 023 000 054 000 048 118 000 126 000 126 118 126 126
126 126 126 126 126 126 126 126 117 126 113 126 117 117 117 113
```

Dotting and Phasing

The two symbols shown as xxx (in the RX position) will be added next. They are part of the "phasing" sequence sent at the start of a message. The phasing sequence lets the decoder find the start of the message, and find the 10-bit word boundaries.

The very start of a message is a "dotting pattern" – a series of alternating 1s and 0s to allow the receiver to synchronize with the bit-rate of the message. The dotting pattern is generally 200 bits long, except for ACK messages of "Selective

Calls", where the dotting pattern is only 20 bits. The longer 200 bit sequence is to allow scanning receivers to find a transmission while scanning several MF/HF channels.

After the Dotting Pattern comes a set of symbols called the "phasing sequence". The sequence of DX and RX characters is

`125 111 125 110 125 109 125 108 125 107 125 106`

before the message data itself begins to be transmitted.

The last two phasing symbols (105 and 104) now interleave with the real message symbols:

`120 105 120 104 000 120 023 120 020…..` etc.

The receiver takes in the bits one at a time, and looks for the pattern "125 109 125 108" etc. by shifting the bits along one at a time until the phasing pattern is found. There are several chances to find the phasing symbols. **Once any three symbols from the phasing sequence** are detected we will be correctly "locked" to the word boundaries, and will be able to count off 10 bits at a time, and treat each 10-bit chunk as a "parity protected word", for processing. This involves parity checking, DX/RX de-interleaving, ECC checking, message parsing etc.

The full time-interleaved message is now (DX and RX)

```
125 111 125 110 125 109 125 108 125 107 125 106 120 105 120 104 000
120 023 120 020 000 000 023 010 020 108 000 023 010 054 108 048 023
000 054 000 048 118 000 126 000 126 118 126 126 126 126 126 126 126
126 126 126 117 126 113 126 117 117 117 113
```

The symbols now need to be converted to 10-bit parity protected words and then we'll have a stream of 62 words x 10 bits – 620 bits. Add on the 200 bit dotting pattern we have 820 bits. At 100 bits per second, a DSC message on MF/HF takes approximately 8 seconds to transmit. Other message formats might be longer or shorter than this "Test" message.

Modulation and transmission characteristics

DSC on the MF and HF bands is transmitted as a Frequency Shift Keyed (FSK) signal at 100 baud. The frequency shift is 170Hz and has the emission code F1B (if direct FSK modulation is used) or J2B if a modulating subcarrier is used in an SSB transmitter. It is usual for J2B transmission to be done with a modem centre frequency of 1700Hz, with the tone representing a binary 1 being the lower of the two transmitted tones. The ITU describes a logical 1 as "Y" and a logical 0 as "B".

The tone frequencies generated in the DSC Modem will therefore be

`Y (1) = 1615Hz        B (0) = 1785Hz`

Recap - Error detection and methods to improve reliability

To improve the successful reception of messages, and to provide a means of detecting errors, DSC uses three methods.

1) Parity checking in each transmitted symbol

2) Repeat transmission of each symbol. They are sent again after four other characters, so each symbol is sent a second time after 400ms have elapsed, which means a burst of noise, or interference, must be longer than 400ms before it can destroy both copies of the same symbol, and we only need one copy to be received correctly to construct the received message.

3) An overall Check Sum test to detect if any symbols have been received in error, even if they passed the initial Parity Check.

Part Two : Analysis of received messages

An error free example – A Test Call

Let’s inspect a received message using only the “RAW” symbols after de-interleaving.

```
120 120 000 023 020 000 070 108 027 033 018 094 000 118 126 126 126
126 126 126 126 117 090 117 117
```

This message has no errors or corruptions and will illustrate how to read the symbols and convert them into a readable message.

Identify the key parts of the message

```
120 120 000 023 020 000 070 108 027 033 018 094 000 118 126 126 126
126 126 126 126 117 090 117 117
```

We know that a message is composed of several distinct sections.

Format = **120**

Called Station MMSI = **000 023 020 000 070**

Category = **108**

Calling Station MMSI = **027 033 018 094 000**

Telecommands 1 & 2 = **118 126**

Message Data (Frequency/Position etc.) = **126 126 126 126 126 126**

EOS = **117**

ECC = **090**

We can convert the MMSI symbols back to the actual 9-digit MMSI:

The “Called Station” is: **000 023 020 000 070** = **002320007**

The “Calling Station” is**: 027 033 018 094 000** = **273318940**

A Format of **120** means “**Individual Stations**”

The Category of **108** means “**Safety**”

Telecommand 1 of **118** means **“Test”**

Telecommand 2 of **126** means **“No Information”**

There are 6 characters for the “message” and here they are all **126** which again means **“No Information”**

The End of Sequence value of **117** means "**REQ**" (Acknowledgement is **REQ**uired)

We have the deciphered meaning of the message

Individual Station Call

To : 002320007

From : 273318940

Safety

Test / No Info

Message : No Information

Acknowledgement Required

This is a simple, very commonly seen, Test message sent from a Ship to a Coast Station.

YaDD logs the message as

FORMAT: SEL *(SEL meaning "Selective Call to an individual station")*

CAT: SAF

TO: COAST,002320007,ENG,Humber Radio

FROM: SHIP,273318940

TC2: NO INFO

FREQ: --

POS: --

EOS: REQ

The "**Message**" field can hold Frequency or Position information, and since the transmitted data in this section was "**126 126 126 126 126 126**" YaDD shows **"FREQ: --"**and **"POS: --"**

Checking the message Error Check Character manually

One final piece of information needs to be dealt with:

ECC = **090**

We know how to calculate the ECC from earlier, and YaDD does this itself, and then compares its calculation with the ECC value from the message. YaDD will show the result of this calculation and comparison:

cECC: 90 OK

We take the basic message, removing the duplicated Format and EOS symbols:

120 000 023 020 000 070 108 027 033 018 094 000 118 126 126 126 126
126 126 126 117

Convert each symbol to binary and do the "XOR" routine on each "column" (bit position).

```
120 = 1 1 1 1 0 0 0

000 = 0 0 0 0 0 0 0

023 = 0 0 1 0 1 1 1

020 = 0 0 1 0 1 0 0

000 = 0 0 0 0 0 0 0

070 = 1 0 0 0 1 1 0

108 = 1 1 0 1 1 0 0

027 = 0 0 1 1 0 1 1

033 = 0 1 0 0 0 0 1

018 = 0 0 1 0 0 1 0

094 = 1 0 1 1 1 1 0

000 = 0 0 0 0 0 0 0

118 = 1 1 1 0 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

126 = 1 1 1 1 1 1 0

117 = 1 1 1 0 1 0 1

      = = = = = = =
```

ECC = 1 0 1 1 0 1 0 = 90

Our calculated ECC matches the received ECC.

We can agree with YaDD : **"cECC = 90 OK"**

We've taken a sequence of 3-figure numbers and converted them into a readable DSC message, we know who sent it, where it was intended for, what type of message it was, and also that we received it with no errors.

Another error free example – not a "Test" call

120 120 023 076 011 000 000 108 023 076 041 000 000 109 126 004 014 090 004 014 090 117 080 117 117

Identify the key sections

120 120 023 076 011 000 000 108 023 076 041 000 000 109 126 004 014 090 004 014 090 117 080 117 117

Format = **120** "Individual Call"

Called Station MMSI = **023 076 011 000 000 = 237611000**

Category = **108** "Safety"

Calling Station MMSI = **023 076 041 000 000 = 237641000**

Telecommand 1 = **109** "J3E Telephony"

Telecommand 2 = **126** "No Information"

Message Data = **004 014 090 004 014 090 = 04149.0kHz / 04149.0kHz**

EOS = **117** "Acknowledgement REQuired"

ECC = **080**

The complete message reads:

```
Individual Station Call

Safety

To: 237611000

From: 237641000

TC1/2: J3E Telephony / No Info

Freq: 4149.0kHz/4149.0kHz

REQ
```

This is a message from a ship with MMSI 237641000, addressed to another ship with MMSI 237611000 requesting that they use SSB Telephony, on 4149kHz. The caller wants a DSC ACK to confirm reception and to confirm the choice of frequency.

What about the ECC?

120 = 1 1 1 1 0 0 0

023 = 0 0 1 0 1 1 1

076 = 1 0 0 1 1 0 0

011 = 0 0 0 1 0 1 1

000 = 0 0 0 0 0 0 0

000 = 0 0 0 0 0 0 0

108 = 1 1 0 1 1 0 0

023 = 0 0 1 0 1 1 1

076 = 1 0 0 1 1 0 0

041 = 0 1 0 1 0 0 1

000 = 0 0 0 0 0 0 0

000 = 0 0 0 0 0 0 0

109 = 1 1 0 1 1 0 1

126 = 1 1 1 1 1 1 0

004 = 0 0 0 0 1 0 0

014 = 0 0 0 1 1 1 0

090 = 1 0 1 1 0 1 0

004 = 0 0 0 0 1 0 0

014 = 0 0 0 1 1 1 0

090 = 1 0 1 1 0 1 0

117 = 1 1 1 0 1 0 1

= = = = = = =

ECC = 1 0 1 0 0 0 0 = 80

Success! Our cECC is 80, which matches the received ECC – the message has no detectable errors.

YaDD logs the message:

```
FORMAT: SEL

CAT: SAF

TO: SHIP,237611000

FROM: SHIP,237641000

TC1: J3E TP

TC2: NO INFO

FREQ: 04149.0/04149.0KHz

POS: --

EOS: REQ

cECC: 80 OK
```

The MMSI of the Sender : 237641000 has an MID of 237. This belongs to Greece, and the vessel is the Knossos Palace. The addressee 237611000 also has an MID of 237 – also a Greek vessel.

Analyzing "Faulty Messages"

One missing symbol

120 120 035 075 063 000 000 108 063 060 015 004 040 ~~~ 126 126 126 126 126 126 126 117 **030** 117 117

YaDD Logs the message

FORMAT: SEL

CAT: SAF

TO: SHIP,357563000

FROM: SHIP,636015044

TC1: UNK/ERR

TC2: NO INFO

FREQ: --

POS: --

EOS: REQ

cECC: 104 ERR

The ECC Checksum test has failed , "cECC: 104 ERR" and it's clear that the Telecommand 1 is showing as "Unk/Err" (Unknown/Error).

What has happened?

Looking at the raw symbols:

120 120 035 075 063 000 000 108 063 060 015 004 040 ~~~ 126 126 126 126 126 126 126 117 030 117 117

We can see that the Telecommand 1 symbol is "missing", it's shown as **~~~,** which means that it failed the ***parity test*** – in fact BOTH the DX and RX copies must have failed the parity test, and we're left with a hole in our message.

The ECC check failed, because the successive XOR of the symbols can't possibly match the correct value, as there's one number missing from the calculation.

YaDD calculates the ECC to be "104" and the ECC we received in the message is "030".

Is the rest of the message ok?

Can we use our knowledge of the structure of DSC messages, and of the ECC calculation, to find a likely value for the missing Telecommand 1? Can we then test our substitution ***with a new ECC calculation***?

Guess the missing symbol and test the solution

Looking at the message, there doesn't appear to be any "Frequency" or "Position" data within the "Message" portion. All the symbols there are **126 126 126 126 126 126** which mean "No Information".

The Format appears to be an "Individual Call" with value **120**

The Category appears to be "Safety" with value **108**

The "End Of Sequence" appears to be a "REQ" with value **117**

We've seen this type of message before, lots of times. It looks like a standard DSC TEST call.

The Telecommand 1 value for "Test" is **118.**

Let's substitute the value of **118** for the missing TC1 symbol and recalculate the ECC.

120 035 075 063 000 000 108 063 060 015 004 040 118 126 126 126 126 126 126 126 117

I get the answer "**Binary 0011110 / Decimal 30"** from my trusty Android Calculator app.

This matches the received ECC in the message!

A repaired message – now error free

We have now shown that one possible version of the message could have been

120 120 035 075 063 000 000 108 063 060 015 004 040 118 126 126 126 126 126 126 126 117 030 117 117

Format : Individual Call

Category : Safety

Called MMSI : 357563000

Calling MMSI : 636015044

Telecommand 1 : **TEST**

Telecommand 2 : No Info

Freq : no info

Position : no info

EOS : REQ

cECC : **30 OK**

If the missing TC1 symbol was the only error, and the correct symbol *HAD* been 118 (for TEST) then the rest of the message would meet the ECC Checksum Test and we are happy that the message is complete and genuine.

An ECC Error, but no missing symbols

```
SYMB: 120 120 035 043 070 000 000 108 000 023 020 020 040 118 126 126
126 126 126 126 126 122 027 122 122

FMT:SEL
CAT:SAF
TO:SHIP,354370000
FR:COAST,002320204,ENG,Snargate Radio Dover
TC1:TEST
TC2:NO INFO
FREQ:--
POS: --
EOS: ACK
ECC: 23 ERR
```

Yadd thinks the ECC has failed – it calculates 23, but has received an ECC symbol with value 27.

There don't *seem* to be any errors though – none of the symbols have failed the **parity test**.

120 035 043 070 000 000 108 000 023 020 020 040 118 126 126 126 126 126 126 126 122

What's the ECC? YaDD says **23**.

My calculator also says **23**.

Why does the message seem to have sent the ECC as **027**?

This symbol passed the **10/7-bit parity test….**

Think of a possible cause of the mismatch in ECC….

"What if the whole message is correct **APART** from the ECC symbol?"

In that case YaDD's (and my) calculated ECC would be correct, the ECC *should be* 23

From the message symbols that we think are correct we've calculated the ECC as Decimal 23 = binary **0010111** . Our supposition is that this may also have been the original "transmitted" ECC.

Has the ECC value been "damaged in transit"?

Let's build a 10-bit parity-protected word from this 7-bit value.

Reverse the bit-order

1110100

Count the zeros : 3

Work out the parity bits: decimal **3** = binary **011**

10-bit parity-protected word is **1110100011**

Now we can look at the received ECC symbol : **27**. What 10 bit word did YaDD's decoder detect, which ***passed the 10/7-bit parity test***, and gave the decoded value of **27**?

Decimal 27 = binary **0011011**.

Reverse the bit-order

1101100

count the zeros: 3

Work out the parity bits: decimal **3** = binary **011**

The 10-bit parity-protected word that represents 27: **1101100011**

This is the 10-bit word that came out of YaDD's decoder, and which met the ***10/7-bit parity test*** before being converted to the decimal symbol "**027**".

Compare the "received" and "calculated" ECCs:

Received 27 (in error?): **1101100011**

Calculated 23 (possibly the true ECC?): **1110100011**

There are only two bits different. Is this an easy glitch to imagine?

If the 10-bit character (for the "true" ECC of **23**) was transmitted as **1110100011** and bits 3 and 4 got "flipped" due to noise while being decoded, the result would be **1101100011** which is still a perfectly valid 10-bit parity-protected word, and when converted back to 7-bits it becomes the decimal number **27** in our decoded message.

Can we really say that the rest of the message is okay?

If we accept that only one symbol is faulty, and that the faulty symbol is the ECC symbol – it is quite easy to accept that we probably do have an accurate decode of the rest of the message.

Here's the proof:

The original message we've just dissected, received at 13:10:21

2013-11-13 13:10:21> 8414.5: 120 120 035 043 070 000 000 108 000 023 020 020 040 118 126 126 126 126 126 126 126 122 **027** 122 122

FORMAT: SEL

CAT: SAF

TO: SHIP,354370000

FROM: COAST,002320204,ENG,Snargate Radio Dover

TC1: TEST

TC2: NO INFO

FREQ: --

POS: --

EOS: ACK

cECC: 23 ERR

By a strange co-incidence the sender repeated his transmission one minute later at 13:11:23

2013-11-13 13:11:23> 8414.5: 120 120 035 043 070 000 000 108 000 023 020 020 040 118 126 126 126 126 126 126 126 122 **023** 122 122

FORMAT: SEL

CAT: SAF

TO: SHIP,354370000

FROM: COAST,002320204,ENG,Snargate Radio Dover

TC1: TEST

TC2: NO INFO

FREQ: --

POS: --

EOS: ACK

cECC: 23 OK

The second message is identical – the same ACK sent to the same vessel. This time YaDD managed to decode the received ECC symbol as **23** and still calculated (from the rest of the received symbols) a value of **23**. The ECCs now match – and we can assume the first message was okay apart from a falsely decoded ECC symbol.

One missing symbol – calculating the likely value

```
08:43:59> 2187.5: 102 102 005 050 001 004 006 108 000 023 020 000
~~~ 109 126 001 092 050 001 092 050 127 023 127 127
           FORMAT: AREA CALL
              CAT: SAFETY
               TO: 55°N=>04° 001°E=>06°
             FROM: COAST 00232000~, UNID
              TC1: J3E TP
              TC2: NO INFO
             FREQ: 01925.0/01925.0KHz
              POS: --
              EOS: EOS
             cECC: 81 ERROR
```

In a previous example we had a message that failed the ECC check due to a missing symbol – a Telecommand – and we found that we could substitute our best guess, and found happily that the ECC check now worked and we declared a successful decode of the message.

In this next example the problem is much the same – **but** the missing symbol this time is an **IMPORTANT** one – it's the last symbol of the coast station's MMSI – the key to identifying the sender of the message.

Can we "fix" our broken message and claim a "catch"?

YaDD calculates the ECC to be **81** using the symbols it has available:

102 005 050 001 004 006 108 000 023 020 000 109 126 001 092 050 001 092 050 127

The message contains an ECC symbol of **23**.

Using the ECC XOR calculation to find the missing value

If we assume no other errors – and that the received ECC of **23** accurately describes the original message, then we can say “if **81** is the XOR value of all the symbols we ***have*** received, and **XXX** is the value of the missing symbol, then if we XOR 81 with the missing value we would HOPE to get an answer of **23**, to match the ECC we received in the message.”

81 xor XXX = 23

81 is the XOR value of all the symbols ***except*** the missing one (which we call **XXX**).

If we’d been creating the ECC value at the transmitter we’d have XORed all the symbols to arrive at 23, but we’ve only been able to XOR ***most*** of them and arrived at 81.

Let’s write out the XOR calculation:

081 = 1 0 1 0 0 0 1

xxx = a b c d e f g

= = = = = = =

023 = 0 0 1 0 1 1 1

Using our knowledge of the XOR truth table can we work out what the values of the x in each column would need to be to make the calculation work?

A	B	XOR
0	0	0
1	0	1
0	1	1
1	1	0

```
1 xor a = 0      a must be 1

0 xor b = 0      b must be 0

1 xor c = 1      c must be 0

0 xor d = 0      d must be 0

0 xor e = 1      e must be 1

0 xor f = 1      f must be 1
```

```
1 xor g = 1      g must be 0
```

Our value for XXX must be binary **1000110** which is decimal **70.**

We've worked out the missing symbol by doing the longhand XOR, *in reverse*, on the binary values of the symbols. Could we use our calculator app to work it out directly?

Aside – manipulating XOR calculations

We want to find the value of **XXX** in the following formula

81 xor XXX = 23

Can we do some "algebra" using the XOR operator so that we can use a calculator instead of writing out all the "ones and noughts"?

We know, from above, that the missing value which satisfies the calculation is actually **70**

81 xor 70 = 23

```
081 = 1 0 1 0 0 0 1

070 = 1 0 0 0 1 1 0

      = = = = = = =

023 = 0 0 1 0 1 1 1
```

What is: **81 xor 23 = ???**

```
081 = 1 0 1 0 0 0 1

023 = 0 0 1 0 1 1 1

      = = = = = = =

070 = 1 0 0 0 1 1 0
```

So… **81 xor 23 = 70**

It turns out that it doesn't matter which way you do the calculation:

81 xor 70 = 23

81 xor 23 = 70

70 xor 23 = 81

As long as we have two values we can work out the missing one.

To reiterate what we've just done. We knew the ECC received in the message was **23** and we knew that our cumulative XOR of the symbols ***that we knew***

about was **81**, and we needed to find out the value of the single missing symbol. By XORing **81** with the wanted **23**, we calculated that the value of the missing symbol had to be **70**.

81 xor 23 = 70

We can use this “XORing works in any order” trick to find missing symbols, saving us the hassle of writing down the “ones and zeros” – and another example will come along soon.

So, we now have a complete set of symbols:

102 102 005 050 001 004 006 108 000 023 020 000 **070** 109 126 001 092 050 001 092 050 127 **023** 127 127

We know now that **cECC** is **23** ***when we make our substitution*** of **070** for the missing MMSI symbol, and that this now matches the received ECC from the message.

So, after all that: **“What is the missing MMSI?”**

YaDD reported : `FROM: COAST 00232000~, UNID`

We can put our newly calculated **070** in place of the final missing symbol:

000 023 020 000 **070**

We know how to retrieve an MMSI from the 3-figure DSC symbols:

0**00** 0**23** 0**20** 0**00** **070**

The MMSI of the UNID Coast Station is : **002320007**

This belongs to **Humber Coastguard** in the UK. Can we now add Humber to our log? After all we received almost all of the message correctly, and by using our knowledge of DSC messages, the ECC calculation and how to manipulate symbols using the XOR function, we have shown that in all probability we must have received:

```
08:43:59> 2187.5: 102 102 005 050 001 004 006 108 000 023 020 000
070 109 126 001 092 050 001 092 050 127 023 127 127

           FORMAT: AREA CALL

              CAT: SAFETY

               TO: 55°N=>04° 001°E=>06°

             FROM: COAST 002320007,ENG,Humber Radio
```

```
 TC1: J3E TP

 TC2: NO INFO

FREQ: 01925.0/01925.0KHz

 POS: --

 EOS: EOS
```

cECC: **23** OK

Some more compelling information? The DSC message is an “Area Call” addressed to vessels in a geographical area . The area being referenced is a box with the co-ordinates:

```
55°N 1°E . . . . . 55°N 7°E
.                         .

.                         .

51°N 1°E . . . . . 51°N 7°E
```

This puts us in the southern North Sea, in Humber’s area of responsibility. The J3E TP frequency is 1925kHz which is one of Humber’s usual MF frequencies for MSI (Maritime Safety Information) broadcasts.

Would we dare to claim a successful reception of Humber Coastguard Radio 002320007 ?

This is a question for each of us to answer ourselves.

We received 24 of the 25 message symbols – only one was missing. The fact that it was part of the sender’s identity is significant, but there’s enough data to allow us to intelligently re-create the missing data.

In this instance, perhaps it doesn’t matter, as Humber sent the same DSC Call **THREE** times – and only the second transmission had the error that we’ve just worked through… the other two transmissions confirm that our calculations were correct though!

```
08:43:50> 2187.5: 102 102 005 050 001 004 006 108 000 023 020 000 070 109 126 001
092 050 001 092 050 127 023 127 127

FORMAT: AREA CALL

   CAT: SAFETY

    TO: 55°N=>04° 001°E=>06°

  FROM: COAST 002320007,ENG,Humber Radio

   TC1: J3E TP
```

```
            TC2: NO INFO
           FREQ: 01925.0/01925.0KHz
            POS: --
            EOS: EOS
           cECC: 23 OK
```

```
08:43:59> 2187.5: 102 102 005 050 001 004 006 108 000 023 020 000 ~~~ 109 126 001
092 050 001 092 050 127 023 127 127
         FORMAT: AREA CALL
            CAT: SAFETY
             TO: 55°N=>04° 001°E=>06°
           FROM: COAST 00232000~, UNID
            TC1: J3E TP
            TC2: NO INFO
           FREQ: 01925.0/01925.0KHz
            POS: --
            EOS: EOS
           cECC: 81 ERROR
```

```
08:44:10> 2187.5: 102 102 005 050 001 004 006 108 000 023 020 000 070 109 126 001
092 050 001 092 050 127 023 127 127
         FORMAT: AREA CALL
            CAT: SAFETY
             TO: 55°N=>04° 001°E=>06°
           FROM: COAST 002320007,ENG,Humber Radio
            TC1: J3E TP
            TC2: NO INFO
           FREQ: 01925.0/01925.0KHz
            POS: --
            EOS: EOS
           cECC: 23 OK
```

The missing ZERO conundrum…

```
20:16:18> 120 120 027 010 002 045 060 108 000 027 011 ~~~ 000 118 126
126 126 126 126 126 126 122 116 122 ~~~
       FORMAT: SELECTIVE CALL
          CAT: SAFETY
           TO: SHIP  271002456
         FROM: COAST 002711~~0, UNID
          TC1: TEST
          TC2: NO INFO
         FREQ: --
         POS : --
          EOS: ACK
         cECC: 116 OK
```

The ECC is "OK" but there's a missing symbol from the Sender's MMSI. How can that be?

120 027 010 002 045 060 108 000 027 011 ~~~ 000 118 126
126 126 126 126 126 126 122

Calculating the ECC from the symbols we've received :

```
120 = 1 1 1 1 0 0 0
027 = 0 0 1 1 0 1 1
010 = 0 0 0 1 0 1 0
002 = 0 0 0 0 0 1 0
045 = 0 1 0 1 1 0 1
060 = 0 1 1 1 1 0 0
108 = 1 1 0 1 1 0 0
000 = 0 0 0 0 0 0 0
027 = 0 0 1 1 0 1 1
011 = 0 0 0 1 0 1 1
~~~
000 = 0 0 0 0 0 0 0
118 = 1 1 1 0 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
126 = 1 1 1 1 1 1 0
122 = 1 1 1 1 0 1 0
      = = = = = = =
ECC = 1 1 1 0 1 0 0 = 116
```

This matches the received ECC value in the message - so our message seems to have no errors**, but what's the MMSI of the Coast Station?** We're missing some data yet and the ECC is "OK"!

The missing symbol (**XXX**) must be a number such that if it was included in the list of numbers in the ECC XOR calculation it would give the same answer **116.**

If we take the answer we've got, so far (**116**), which was obtained by calculating the ECC and *ignoring the missing number*, and then we XOR this answer with the true value (**XXX**) of the missing number, we will arrive at the TRUE ECC. We believe this TRUE ECC to be **116** since that's the value we've taken from the message itself. What number, XORed with **116** gives an answer of **116**?

116 XOR XXX = 116

```
116 = 1 1 1 0 1 0 0
XXX = a b c d e f g
      = = = = = = =
116 = 1 1 1 0 1 0 0
```

A	B	XOR
0	0	0
1	0	1
0	1	1
1	1	0

```
1 xor a = 1      a must be 0

1 xor b = 1      b must be 0

1 xor c = 1      c must be 0

0 xor d = 0      d must be 0

1 xor e = 1      e must be 0

0 xor f = 0      f must be 0

0 xor g = 0      g must be 0
```

The missing symbol (XXX) must have been binary **0000000**

We can do our "XOR manipulation" from the last example, where we discovered that

A xor B = C

B xor C = A

A xor C = B

So can we just calculate the missing number, and not mess about with the “ones and noughts”?

116 XOR XXX = 116 can be re-written **116 XOR 116 = XXX** and our calculator tells us that **116 XOR 116 = 0**

Assuming NO OTHER ERRORS……

The missing symbol must have been **000**, and had it been received correctly the overall ECC calculation would have given the same result…. **116**

The missing MMSI is: **000 027 011 000 000**

002711000 is Istanbul Radio

The message should look like:

```
20:16:18> 120 120 027 010 002 045 060 108 000 027 011 000 000 118 126
126 126 126 126 126 126 122 116 122 ~~~
       FORMAT: SELECTIVE CALL
          CAT: SAFETY
           TO: SHIP  271002456
         FROM: COAST 002711000,TUR,Istanbul Radio
          TC1: TEST
          TC2: NO INFO
         FREQ: --
         POS : --
          EOS: ACK
         cECC: 116 OK
```

Do we log this as a successful catch of Istanbul?

What we've found is that if we have a single missing symbol, yet the received ECC matches the calculated ECC, then the missing symbol must have been "000". Zero does not change the cumulative XOR result.

One last DX Debunking

Is this reception really Honolulu on 2MHz?

RX:2187.5

SYMB: 120 120 031 094 068 000 000 108 000 036 069 099 034 118 ~~~ 126 126 126 126 126 126 122 049 ~~~ ~~~

FMT:SEL

CAT:SAF

TO:SHIP,319468000

FR:COAST,003669993,HWA,CAMSPAC Honolulu

TC1:TEST

TC2:UNK/ERR

FREQ:--

POS: --

EOS: ACK

ECC: 61 ERR

Take the raw message and highlight the sections as usual

120 031 094 068 000 000 108 000 036 069 099 034 118 ~~~ 126 126 126 126 126 126 122 049

Replace the missing symbol and re-test the ECC

My first thought – the ECC fails but is this simply due to the "missing Telecommand 2" symbol – and shouldn't it be **126**?

Will this substitution of **126** fix the ECC and can we then claim a really good DX catch?

Let's try… The cECC using all our received symbols (i.e. except the missing TC2 symbol) is **61**.

The new cECC, including the extra **126** is easy enough to work out :

61 xor 126 = 67

That still doesn't equal the received ECC of **49**. So, there's another error!

We'll keep the new Telecommand 2 value of **126**, because that "***just seems right***".

Find the error

Let's look at the MMSI symbols which we've received, which pointed us initially to the MMSI for Honolulu **003669993** – the rare 2MHz DX. You have to be suspicious.

000 036 069 099 034

We ***know*** how to convert between MMSI and the DSC symbols and vice versa – it's about "padding with zeros".

000 036 069 099 034

The padding zeros have been **highlighted** – but wait!

One of them is a 4 – that can't be right.

The DSC transmitter takes a 9-digit MMSI and puts those digits into 5 DSC symbols – and the last symbol ALWAYS has a zero at the end, **ALWAYS, ALWAYS!**

The last symbol is **WRONG** – and it may not have been Honolulu after all. (Who'd have guessed?)

A dead end

But wait **(again)** – what if the last symbol was really been **030**. Then it still could have been Honolulu.

000 036 069 099 030

Recalculate the ECC but use **030** instead of **034** – this gives us **127**! (Go on, try it yourself…)

Not the **47** we wanted.

Back on track

Okay – we'll discard the last MMSI symbol (**034**) completely, and calculate an ECC without it….

New cECC (of all symbols except the now discarded **034**) = **97**

What missing symbol (the final missing MMSI symbol) would XOR with **97**(the cumulative XOR of all the symbols we believe to be correct) and arrive at the all important (true?) ECC of **49**?

97 xor **XXX** = **49**

we know from previous examples that this can be written as

97 xor 49 = XXX

our calculator tells us that

97 xor 49 = 80

The missing symbol from the end of the Coast Station MMSI seems to be **080** (this ends in a zero too, which is good!)

The MMSI is therefore **000 036 069 099 080**

000 036 069 099 080

003669998 is "COMMSTA New Orleans" which is a much more likely 2MHz catch, than Honolulu, here in the UK.

The solution to our "two error" problem

Repairing all the damage, the full message should have been:

120 120 031 094 068 000 000 108 000 036 069 099 **080** 118 **126** 126 126 126 126 126 126 122 **049**

FMT:SEL

CAT:SAF

TO:SHIP,319468000

FR:COAST,**003669998,USA,COMMSTA New Orleans**

TC1:TEST

TC2:No Info

FREQ:--

POS: --

EOS: ACK

ECC: 49 OK

Confidence check….

The vessel in the message **319468000** is the "Stolt Confidence" an Oil Tanker, and at the time of the message she was in the Gulf of Mexico, off shore from Houston, and very close to New Orleans. It all seems to point to a positive ID of New Orleans, and not Honolulu.

How did YaDD get the last symbol of the MMSI wrong?

We initially received the last symbol of the MMSI as **034**, but now we are confident the real symbol was **080**.

Is this a "false positive parity" error?

If the "real" symbol was **080** – then what was the 10-bit word that the transmitter sent?

Decimal **80** = Binary **1010000**

Reverse the order : **0000101**

Count the zeros **5**

Parity bits **5** = binary **101**

The 10-bit word representing a symbol of **80** : **0000101101**

YaDD decoded the symbol as **034**- so what was the 10-bit word that must have been presented, which passed the parity test?

Decimal **34** = Binary **0100010**

Reverse the order : **0100010**

Count the zeros **5**

Parity bits **5** = binary **101**

The 10-bit word representing a symbol of **34** : **0100010101**

Compare the two 10-bit words:

80 (the real symbol): **0000101101**

34 (the false positive?): **0100010101**

There are **FOUR** *bit errors* in the "wrong" value of **34**comparing it with the "right" value of **80**. This is surprising, but goes to show how badly corrupted a 10-bit word can become, and still end up with a valid "***parity test***".

Logging and Reporting corrupted messages

What constitutes an acceptable “repair”?

Here is a basic sequence that I would recommend, when faced with a message that has some form of error. The error might be that the ECC check has failed or that there are missing symbols, even when the ECC check is “OK”.

If the ECC has failed (the most common problem) can we see an obvious reason?

1) Missing symbol(s)?

In the case of missing symbol(s)

a) Missing symbol from a predictable element of the message?

b) Missing symbol from the MMSI?

c) Missing ECC symbol?

2) Obvious error in a symbol that has a predictable value?

An error in a “predictable” symbol would be something like

a) A “null data” symbol 126 in error

b) A Telecommand One or Telecommand Two symbol in error.

3) Error in a symbol that is not immediately obvious

a) MMSI symbol in error

b) Frequency or Position symbols in error

c) ECC symbol in error.

The easiest to handle are

1) "missing 126" symbols:

```
TIME: 2013-11-25 09:33:58 FREQ: 2187.5

SYMB: 120 120 025 078 025 000 000 108 000 025 070 050 000 118 126 ~~~ 126
126 126 126 126 122 069 122 122

 FMT: SEL

 CAT: SAF

  TO: SHIP,257825000

FROM: COAST,002570500,NOR,Floroe Radio

 TC1: TEST

 TC2: NO INFO

FREQ: --

 POS: --

 EOS: ACK

cECC: 59 ERR
```

Replacing the ~~~ with 126 and recalculate the cECC (simply the original cECC of 59 xor'd with the replacement "126")

59 xor **126** = **69**

The message is now valid and only one minor error to correct.

2) “missing Telecommand One or Two”

```
TIME: 2013-11-25 02:08:23 FREQ: 2187.5

SYMB: 120 120 000 021 091 000 000 108 025 077 086 000 000 ~~~ 126 126 126
126 126 126 126 117 037 117 117

 FMT: SEL

 CAT: SAF

  TO: COAST,002191000,DNK,Lyngby Radio

FROM: SHIP,257786000

 TC1: UNK/ERR

 TC2: NO INFO

FREQ: --

 POS: --

 EOS: REQ

cECC: 83 ERR
```

Replace the missing Telecommand One with the expected “118” (meaning “TEST”) and recalculate the cECC.

83 xor **118** = **37**

The message is now valid, and again only one minor error to correct.

3)One Missing symbol – perhaps from the MMSI?

Potentially a bit more risky, but a single missing symbol, assuming no other errors, can be calculated

TIME: 2013-11-25 08:10:42 FREQ: 2187.5

SYMB: 120 120 000 025 070 080 000 108 000 ~~~ 070 080 000 118 126 126 126 126 126 126 126 122 **102** 122 122

FMT: SEL

CAT: SAF

TO: COAST,002570800,NOR,Vardo Radio

FROM: COAST,00~~70800, UNID

TC1: TEST

TC2: NO INFO

FREQ: --

POS: --

EOS: ACK

cECC: **127** ERR

Calculate a missing value by using the received ECC and the cECC symbols:

127 xor **102** = **XXX**

127 xor **102** = **25**

Replace the missing MMSI symbol with "025" and we now have:

000 025 070 080 000

002570800 is Vardo Radio.

Since this is a test call ACK from Vardo, addressed to Vardo, I think it's fair to assume we can accept our correction.