Lindecker Patrick (F6CTE) the 22th of June 2023

Specifications of the EM (Extended Message) mode - Version 3

Comparison of the version 2 with the version 1

The version 2 is not compatible with the version 1 (issued the 10th of November 2018). In fact, the version 2 is compatible with the version 1 in this direction: Version 2 → Version 1 but not in the reverse direction Version 1 → Version 2.
The version 2 EM mode is more reliable and a bit more sensitive at detection level than the version 1 one.
The version 2 is applied on the version 4.45.6 of Multipsk for the EM mode. The former EM sub-mode (under version 1) is deleted from the 4.45.6 Multipsk version.

Improvements of the version 3

The version 3 is first applied on the version 4.48.2 of Multipsk. The version 3 (compatible with the version 2) introduces and describes two types of beacon:

- One transmitting the 6 characters Locator (accuracy +/- 4 km), preceded by the call sign (12 characters max) and followed by the power transmitted (1 mW to 1500 W), the type of antenna plus its directivity and a free text of 26 characters. This beacon is rather aimed to a fixed station. However, thanks to a GPS tracking, the station can be mobile and so can be tracked.

- One transmitting the exact GPS position (+/- 1 m but in fact limited by the GPS accuracy, i.e. about +/- 10 m) plus the altitude, preceded by the call sign and followed by the GPS date/time, the number of satellites received, the speed over ground (in knots), the course over ground (in °), the power transmitted (1 mW to 1500 W), the type of antenna plus its directivity and a free text of 10 characters. This beacon is aimed to a mobile station.

Note that thanks to the Reed Solomon coding and to a checksum, the probability to display a false beacon frame is extremely weak.

SUMMARY

1) Preliminaries Page 3
Use conditions
References
Abbreviations
Frequency management
Main use
Short description

2) Modulation and demodulation Page 5
Modulation description
Demodulation description

3) Making of the text message Page 6

4) EM frame description (emission and reception) Page 7
Detection code
RS sequence
Multi-occurrences of EM

5) Locator and GPS beacons description Page 9
Locator beacon
GPS beacon

APPENDIX A (Accented letters in capitals (for the 6 bits characters) and more Page 15

APPENDIX B (DELPHI 6 procedures and functions called by the main text) Page 16

1) Preliminaries

Use conditions
This protocol is free for use. It is as complete as possible. However, the author does not guarantee neither the exhaustively of the information necessary to make it work, nor the absence of technical errors in the writing.

References
* "Technical aspects of RS ID and Call ID and use" from the author:
http://f6cte.free.fr/PAPERS.ZIP

Abbreviations:
"**EM**" is worth for "Extended Message",
"**RS**" is worth for "Reed Solomon".
"**ID**" is worth for "Identifier" (as in “RS ID”).

Frequency management
The bandwidth of a transmission being equal to 172 Hz, it is considered that a channel is equal to 400 Hz (172 Hz + 228 Hz of margin) and contains 0 or 1 EM QSO. So, for a standard 2.5 kHz bandwidth (200 to 2700 Hz), there are 6 channels: 200 to 600 Hz, 600 to 1000 Hz, 1000 to 1400 Hz, 1400 to 1800 Hz, 1800 to 2200 Hz, and 2200 to 2600 KHz. A channel of 400 Hz is worth for a medium screen (in term of number of pixels). For a small screen, 600 Hz will be necessary, but 200 Hz will only be necessary for a very broad screen.

Now, the maximum theoretical number of QSO would be around 2500/172=14 (172 Hz being the transmission bandwidth).

Example for a 2.5 KHz bandwidth on the waterfall (6 transmissions done in the same time)

```
400 Hz      800 Hz      1200 Hz     1600 Hz     2000 Hz     2400Hz
Sender 1    Sender 2    Sender 3    Sender 4    Sender 5    Sender 6
   |           |           |           |           |           |
CQ F9XYV... CQ F9XYW... CQ F9XYX... CQ F9XYY... CQ F9XYZ    CQ F9XZZ
Addressee 1 Addressee 2 Addressee 3 Addressee 4 Addressee 5 Addressee 6
```

Main use
Its main use is to give the possibility to have simultaneous QSO (at standard speed) between different Hams, as messages are displayed on the waterfall. So everyone can see these messages and possibly answer. The number of QSO is limited only by the bandwidth. A channel of about 400 Hz on the waterfall is adequate to display the message. So 6 QSO in a standard phone bandwidth (2500 Hz) will be nice, each QSO being visible by everybody.

With the EM, it is possible to send messages by blocks of 47 characters maximum each.
A big message will be automatically split in blocks of 47 characters, sent one after the other.
Its use is similar to the one of the FT4 or FT8 modes but at a very superior speed, without any constraint on the transmission time and on the message nature (non formatted), but of course, the sensitivity is not comparable with the FT4 or FT8 one, but rather to the Olivia or BPSK31 one..

Example of an EM « CQ » message displayed on the waterfall.

As PSK31, RTTY, EM is not an ARQ mode, i.e. the sender ignores if his/her message has been received or not by the addressee.

For one EM transmission, the EM frame is composed of 42 to up 108 symbols of 4 bits according to the message length + two symbols at the beginning of the transmission (two repetitions of the first symbol of the detection code) to leave 186 ms for the VOX to switch the transceiver. So it is transmitted in a time between 4.1 and 10.2 sec.
It is possibly to send an automatic RS ID on a CQ call. Manual RSID transmissions are not authorized for this mode, to limit interactions with the possible other QSO in the same AF band. Note that the specific RS ID number for the EM mode is 279.

The EM is always detected and decoded at a signal-to-noise ratio of -13 dB, so with a sensitivity better than PSK31 (and comparable with MFSK16 or slow Olivia). Note that the central frequency of the EM transmission is determined with a precision of +/- 2.7 Hz.

As soon as an EM is detected and decoded, Multipsk displays the message received on the waterfall under the transmission and on reception panels. It is also proposed to the user to switch on the EM transmission frequency.

In presence of an EM frame detection, there are only two possible outputs from the decoding algorithm:

- either it is not decoded (i.e. there are more than 8 errors in the frame) because the signal-to-noise ratio is too bad,
- or it is decoded and it is correct (i.e. no error). The probability of detecting a wrong EM (seen correct) is almost nil.

However, it must exist an option to display (between two "?", for example) the detected but erroneous frames, the channel being too much noisy. This because, even incorrect, these frames can be more or less readable and so must be of interest for the Ham, at least to know that the other Ham has answered.

The maximum speed is equal to 46 wpm (similar to PSK31, MFSK16…), as a message of 47 characters is transmitted in 10.2 seconds (a “word“ is equivalent to 6 characters).

Short description
Baud rate: 10.766
Speed : 46 wpm maximum
Characters set: part of the ASCII (CHR(32) to CHR(95))
Modulation : 16MFSK (4 bits per symbol), no Gray coding
Bandwidth : 172 Hz (note that there is no filtering).
Bit synchronization : automatic using the signal

Data synchronization : thanks to a detection code of 13 symbols transmitted before the Reed Solomon frame. It authorizes 3 symbols in error.
Correction code: Reed Solomon 63-47 of variable length (so possibly shortened) and containing a maximum of 47x6=282 data bits
Convolution code: no
Interleaving : no
Scrambling: no
Lowest S/N: -13 dB.

Reception mode: sensible to the side (USB or LSB), USB is compulsory.

2) Modulation and demodulation

The modulation and the demodulation are similar as the ones used for RS ID.

Modulation description
It is the same modulation as the one used for RS ID.

Each symbol of 4 bits is transmitted using MFSK modulation. There are 16 possibilities of frequencies separated by 11025/1024=10.766 Hz. Each symbol transmission is done on only one frequency for a duration equal to 1024/11025x1000=92.88 ms (so at 10.766 bauds).

Note about the average transmitted power
The ratio between the average power and the maximal power is equal to 1 because the EM is transmitted in MFSK, so the transmitted power is maximum.

Demodulation description

For each semi-step of time (46.44 ms) and for each semi-step of frequency (5.38 Hz), the program attempts to detect a detection code of 13 symbols representative of an Extended Message (EM) with a very low probability of error. So each second, about 10000 possible EM are tested, depending on the selected bandwidth, i.e. 200-2500 Hz, 200-3300 Hz or 200-4300 Hz on Multipsk, the standard being 200-2500 Hz.

The analysis is based on FFT (Fast Fourier transform) of 2048 points at 11025 samples/sec, regularly done at each semi-step of time (46.44 ms).

3) Making of the text message

The RS block can host 47 characters (6 bits/character) as a maximum. For a given message, it will be allowed 3 characters ("L=0"), 6 characters ("L=1"),..., 45 characters ("L=14") or 47 characters ("L=15"), according to the message size.

> Note 1: "L" is a variable (from 0 to 15), used in the program (see §4).
>
> Note 2: on Multipsk, the message typed by the Ham is sent when he/she types the carriage return (if validated) or clicks on the "**Send**" button. He/she can stop the transmission (in case of error) by clicking on the "**Stop**" button.

The 6 bits characters used are part of the ASCII characters (used for occidental languages), from CHR(32) to CHR(95), except:

- The "^" character (used as a "filling" character).
- The "[" character (reserved for the future as a control character). It must not be used.
- The "]" character (used for the Carriage Return + Line Feed).

So a message can contain 47 characters as a maximum. For a character, the value sent is equal to ORD(character)-32. For example, the value of the Space (CHR(32)) is 0 and the value of "_" is 63.
The received characters will be displayed either in capitals (by default) or in small letters. Specifically for the display on the waterfall, the Carriage Return is transformed in a space and possible initial and final spaces are not displayed.
The accented letters (used in Spanish, French…) and some other characters will be transformed in capitals before transmission. See Appendix A.

If the text to send comes from a sequence (macro), it will be transformed so as be to compatible with the 6 bits characters. In that case, the Carriage Return will be internally transformed in the "]" character.

As the text is sent by blocks of 3 characters (see §4)), the remaining room in the frame (one or two 6 bits characters) will be filled with one or two "^" characters, the "^" character ("111110" in binary) being only considered as a filling character.

4) EM frame description (emission and reception)

The EM frame is composed of a detection code of 13 symbols of 4 bits followed by a Reed Solomon sequence of 29 to 95 symbols of 4 bits.

Detection code
The detection code is a pseudo random sequence of 13 symbols of 4 bits each giving the value of the symbol L (used further). Here is the table of the 16 possible sequences (in hexadecimal) according to L:

- L=0: 6EB32109A3886
- L=1: 83743AEBF2D7E,
- L=2: A90B9F4912E0B
- L=3: 00AC238293ED3
- L=4: 789B8DB521799
- L=5: F1430B4678F2B
- L=6: 18AE00A3776E5
- L=7: 953CA8E44424F
- L=8: DB47C08B0D489
- L=9: 27749FF786BCE
- L=10: BCC4AE31B9E89
- L=11: 5E67FCC8D831C
- L=12: 359F8981AB1B4
- L=13: 458A75C0E4857
- L=14: CC3E73F2C7883
- L=15: E18DECF454758

It is enough to receive 10 symbols over the 13 symbols. In other words, it is accepted 3 symbols in error over the 13 symbols.

The detection code has several functions:

- detection of an EM (with a probability of 3 errors of about 4.6 E-9),
- determination of the message length (=(L+1)*3) inside the RS sequence and, consequently the total RS sequence length=16+(L+1)*3 in 6 bits characters.

Taken into account of the VOX delay
The switch to TX is almost instantaneous through a fast relay or through a CAT system. Now it is not the case through a VOX which is relatively slow. So to accommodate this problem, the first symbol of the detection code will be sent three times, which will leave 2x93=186 ms for the VOX to switch the transceiver.

RS (Reed Solomon) sequence
The detection code is followed by the RS sequence which length has been defined by the detection code through L. If the bit length of the RS sequence is not a multiple of 4 (transmission symbols being of 4 bits), it is added the double bit "00" to complete. For example for L=0 the RS length is equal to 16+(L+1)*3=19 characters of 6 bits, so for 19x6=114 bits which is not a multiple of 4, it is added "00" to get 116 bits which is a multiple of 4 (116/4=29 symbols of 4 bits). But for L=15 (max), the RS length is equal to 16+(L+1)*3=64, limited to 63 characters of 6 bits, so for 63x6=378 bits which is not a multiple of 4, it is added "00" to get 380 bits which is a multiple of 4 (380/4=95 symbols of 4 bits). Note that added to the 13 first symbols, the minimum of symbols of 4 bits is equal to 29+13=42 as a minimum and is equal to 95+13=108.

The RS sequence is at the format 63-47, with the power of the first root
(b) at 4. It can be fixed 8 characters of 6 bits as a maximum.
The RS sequence is composed as a maximum by 63x6=378 bits. If the RS sequence is shortened (down to 19 characters), it will be internally considered that the missing (i.e. hidden) characters are spaces (i.e. "000000" in binary).
The 96 parity bits (24 symbols of 4 bits) are sent first, so before the message bits.

For details, see, in Appendix B, this procedure extracted from the Multipsk code:
CODING_OF_THE_EXTENDED_MESSAGE_ID based on the Delphi Unit RS_EM (Appendix B)

Note that in RX, once the RS sequence decoded, the message will be displayed on the waterfall and on the reception panel(s) (to store the previous messages).

Exemples
The EM message for “A” will be coded, in hexadecimal, as:
5DA210/892775C7B0CA6DB8739D83A456D76976CE7

The EM message for “ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTU” will be coded, in hexadecimal, as:
D07CDBE343647ECC705981A/1A94D8D4CA3B47517D38558D79599DBA5DAE/B61BE3C65CE7D697517D38558D79599DBA5DAE/B61BE3C3

Multi-occurrences of EM
Of course, Hams can send EM in the same time. So it will be considered that a maximum of 25 messages can be received in parallel, because “25” messages” can possibility be hosted on a 4300 Hz bandwidth with one EM each 172 Hz. However, 25 is just a limit. More reasonably, as recommended in §1, it must be used channels of 400 Hz.

5) Locator and GPS beacons description

5.1 Generalities

The “[” character is used as a first character to introduce a formatted frame. To code the rest of the formatted frame (Locator beacon and GPS beacon), it must not be used the following characters:

- the “^” character. Now it can be used 0, 1 or 2 “^” just after the formatted frame as filling character (see §3 for details),
- the “[” character,
- and the “]” character (used normally for the Carriage Return + Line Feed).

So only 61 (0..60) characters can be used for the coding. Consequently, all numbers are built with a 61 numerical base. In DELPHI 6 the highest integer number in INT64 format is worth 2^63-1.
This limit applies to code strings and numbers. So 11 EM characters is the maximum which can be used to code some data.

5.2 Locator beacon

It is transmitted first the prefix (two first characters) then the 6 characters Locator (accuracy +/- 4 km), preceded by the call sign (12 characters max) and followed by the power transmitted (1 mW to 1000 W), the type of antenna plus its directivity and a free text of 27 characters.

> Note: the Locator can be fixed for a fixed station, but it is can also be updated by the GPS position in case of a mobile station.

Prefix
The prefix used for “Locator beacon” is the double character “[A”.

Call sign
The length of the Ham call sign is between 3 and 12 characters and it must be only composed of figures, letters and “/”, as for example 1Z2/F6CTE/MM. However, "/" can’t be located at the beginning or at the end of the call sign (this exception could be used in the future). Except these constraints, there is no other check about the validity of the Ham call sign. It is added spaces after the call sign to reach 12 characters. These 12 characters are transformed in a 11 EM characters string. 2 bits are not used. They are reserved for the future. For details, see, in Appendix B:
ADDING_FINAL_SPACES_TO_REACH_THE_RIGHT_LENGTH +
CODING_OF_THE_FREE_HAM_CALL_SIGN_IN_EM_STRING
DECODING_OF_THE_EM_CALL_SIGN_IN_CALL_SIGN + REMOVAL_OF_FINAL_SPACES

Locator 6 characters
The Locator format is first checked.

> Note 1: a Locator 4 characters is transformed in Locator 6 characters adding “MM”.
> Note 2: the Locator can be obtained from the GPS position (latitude/longitude) via a simple conversion. See the §5.3 for more information about the GPS latitude and longitude.

These 6 characters are transformed in a 5 EM characters string. 5 bits are not used. They are reserved for the future. For details, see, in Appendix B:
CHECK_AND_FORMATTING_OF_THE_LOCATOR
CODING_OF_THE_LOCATOR_IN_EM_STRING
DECODING_OF_THE_EM_LOCATOR_IN_LOCATOR

Transmission power:
The transmission power between 1 mW to 1500 W is split into 61 values, i.e. 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9 mW, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90 mW,…, 500 W, 600 W, 700 W, 800 W, 1000 W, 1500 W. The value between 0 and 60 is carried into 1 EM character. For details, see, in Appendix B:
CODING_OF_THE_POWER
DECODING_OF_THE_POWER

Antenna

It is managed two pieces of information about the user antenna:

- The type of antenna or the relative gain with respect to isotropic antenna (same power in all directions). The 5 first types are omnidirectional and don't need a specific directivity. The 7 last being directional antennas, they are associated to a direction.

N	Antenna or gain
0	Vertical Whip antenna (omnidirectionnal)
1	Vertical Helical antenna (omnidirectionnal)
2	Discone antenna (omnidirectionnal)
3	Horizontal Halo antenna (omni directionnal)
4	Horizontal Loop antenna (omni directionnal)
5	Magnetic loop
6	Dipole (gain around 2 dBi)
7	4 dBi (for example Quad loop)
8	7 dBi (for example 2 elements Yagi)
9	10 dBi (for example 2 elements Quad or 3 elements Yagi)
10	13 dBi (for example 4 to 7 elements Yagi or 3 elements Quad)
11	16 dBi and more (for example 5 or more elements Yagi or Quad)

- The directivity of the directional antenna, on 8 values:

N	Directivity
0	North East
1	East
2	South East
3	South
4	South West
5	West
6	North West
7	North

These two pieces of information are carried into 1 EM character. For details, see, in Appendix B:
CODING_OF_THE_ANTENNA_GAIN_AND_DIRECTIVITY
DECODING_OF_THE_ANTENNA_GAIN_AND_DIRECTIVITY

Free text

It is allowed a free text from 0 to 26 characters. This text can be, for example, a name. Look at 3) for details. This free text does not contain any “Carriage Return + Line Feed” (“]” character), nor the “^” or “[” characters. Moreover, “[“ is transformed in "(" whereas “]“ is transformed in ")". There is no coding of the text but a control to check that characters are compatible with the EM set of characters. The control of this free text is given in the procedure in Appendix B: CONTROL_OF_A_FREE_TEXT

Checksum

It is determined a checksum on one EM character with all the previous characters of the EM frame, i.e. from the prefix included to the free text included. Note that thanks to the Reed Solomon coding and to this checksum, the probability to display a false beacon frame is extremely weak. For details, see, in Appendix B: CHECKSUM_EM_FRAME

About the length of the frame

In short, the beacon frame is composed of:

- Prefix: 2 EM characters
- Call sign: 11 EM characters
- Locator 6 characters: 5 EM characters
- Transmission power: 1 EM character
- Antenna: 1 EM character
- Free text: 0 to 26 EM characters
- Checksum: 1 EM character

The minimum length is 21 EM characters (with no free text). The maximum length is 47 EM characters, which is the maximum length for a EM frame.

Exemple

The pieces of information “F6CTE / JN18FT / P=10 W / Magnetic Loop -> SE / PATRICK MAISONS-ALFORT” will be coded in ASCII characters as:
A"2*?=;D%R.& L;O)H'PATRICK MAISONS-ALFORTD

The EM message will be coded, in hexadecimal, as:
BB2D62E1B6772857/0AD4DA6A8A80AA/EC994DD0/8118E64A8/4B7D07/A0AAB89/6B10C219529B/A50962ADDBBC75B89EBA382DE7

5.3 GPS beacon

It is transmitted first the prefix (two first characters) then the exact GPS position (+/- 1 m but in fact limited to the real GPS accuracy, i.e. about +/- 10 m) plus the altitude, preceded by the call sign and followed by the GPS date/time, the number of satellites received, the speed over ground (in knots), the course over ground (in °), the power transmitted (1 mW to 1000 W), the type of antenna plus its directivity and a free text of 10 characters. This beacon is rather aimed to a mobile station.

Note: it is reminded that the position given by the $GPRMC GPS frame is only valid is the checksum is correct and if the status is on “A” (data valid).

Prefix
The prefix used for “GPS beacon” is the double character “[B”.

Call sign
Same as the one described in §5.2.

Latitude
The $GPRMC frame gives the latitude in the form “DDMM.TTT,X” with “D” for degrees, “M” for minutes, “T” for thousandths of minute and X for “N” (North) or “S” (South). The latitude is carried into a 4 EM characters string, according to procedures in Appendix B:
CODING_OF_THE_LATITUDE
DECODING_OF_THE_LATITUDE

Longitude and altitude
The $GPRMC frame gives the longitude in the form “DDDMM.TTT,X” with “D” for degrees, “M” for minutes, “T” for thousandths of minute and X for “E” (East) or “W” (West).
The $GPGGA frame gives the altitude with or without a decimal. It can be added other pieces of information (no data, altitude not validated, too low or two high altitude). The longitude and the altitude are carried into a 7 EM characters string according to the procedures in Appendix B:
CODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE
DECODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE

Speed over ground, number of quarters of a century and number of satellites
The $GPRMC frame gives the speed over ground with or without a decimal and the date. This frame gives also the date. From the date, the year is extracted to give the number of quarters of a century (to determine the year afterwards). The $GPGGA frame gives the number of satellites (3 to 12 normally). These three pieces of data are carried into a 3 EM characters string according to the procedures in Appendix B:

CODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES

DECODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES

Course over ground
The $GPRMC frame gives the course over ground with or without a decimal. This piece of data is carried into a 2 EM characters string according to the procedures in Appendix B:
CODING_OF_THE_COURSE_OVER_GROUND
DECODING_OF_THE_COURSE_OVER_GROUND

GPS UTC date and time:
They are extracted from the GPS $GPRMC frame. The year of the UTC date is coded in the EM frame between 0 and 24. Thanks to the number of quarters of a century (see above “Speed over

ground…), the exact year (0 to 99) is reconstructed. The UTC time is supposed always present, i.e. without any time, the frame is supposed invalid and is not taken into account, whereas the UTC date is supposed optional. Both pieces of data are carried into 5 EM characters. For details, see, in Appendix B:
CODING_OF_THE_TIME_AND_DATE
DECODING_OF_THE_TIME_AND_DATE

Transmission power:
Same as the one described in §5.2.

Antenna
Same as the one described in §5.2.

Free text
It is allowed a free text from 0 to 10 characters. Except the maximum length, the specifications are the same as the ones described about free text in §5.2.

Checksum
It is determined a checksum on one EM character with all the previous characters of the EM frame, i.e. from the prefix included to the free text included. Note that thanks to the Reed Solomon coding and to this checksum, the probability to display a false beacon frame is extremely weak. For details, see, in Appendix B: CHECKSUM_EM_FRAME

About the length of the frame
In short, the beacon frame is composed of:

- Prefix: 2 EM characters
- Call sign: 11 EM characters
- Latitude: 4 EM characters
- Longitude and altitude: 7 EM characters
- Speed over ground and number of satellites: 3 EM characters
- Course over ground: 2 EM characters
- GPS UTC date and time: 5 EM characters
- Transmission power: 1 EM character
- Antenna: 1 EM character
- Free text: 0 to 10 EM characters
- Checksum: 1 EM character

The minimum length is 37 EM characters (with no free text). The maximum length is 47 EM characters, which is the maximum length for a EM frame.

Exemple
The pieces of information “F6CTE / 10 sat. 12:29:35 17/05/2023 48-48.721N 002-25.901E 230 ft V 0.1 knot(s) / 10 W Magnetic Loop -> SE / PATRICK” will be coded in ASCII characters as: B"2*?=;D%R.&DJ29>WEEK_V !'_!V039IH'PATRICK.

The EM message will be coded, in hexadecimal, as:
BB2D62E1B6772C496D862BB2E438A65347767DD1/8118E64A8/4B7D09398154DCD485A
EE5//00EE/653025589/6B10C219529BDE7

APPENDIX A

Accented letters in capitals (for the 6 bits characters) and more

```
IF (THE_CHAR='Ç') OR (THE_CHAR='ç') THEN THE_CHAR:='C' ELSE

IF (THE_CHAR='ü') OR (THE_CHAR='û') OR (THE_CHAR='ù') OR
   (THE_CHAR='Ü') OR (THE_CHAR='ú')  THEN THE_CHAR:='U' ELSE

IF (THE_CHAR='é') OR (THE_CHAR='ê') OR
   (THE_CHAR='æ') OR (THE_CHAR='Æ') OR (THE_CHAR='ë') OR
   (THE_CHAR='è') OR (THE_CHAR='É')  THEN THE_CHAR:='E' ELSE

IF (THE_CHAR='â') OR (THE_CHAR='ä') OR (THE_CHAR='à') OR
   (THE_CHAR='å') OR (THE_CHAR='Ä') OR (THE_CHAR='Å')
   OR (THE_CHAR='á')  OR (THE_CHAR='ª') THEN THE_CHAR:='A' ELSE

IF (THE_CHAR='ï') OR (THE_CHAR='î') OR (THE_CHAR='ì') OR
   (THE_CHAR='í') THEN THE_CHAR:='I' ELSE

IF (THE_CHAR='ô') OR (THE_CHAR='ö') OR (THE_CHAR='ò') OR
   (THE_CHAR='Ö') OR (THE_CHAR='ó') OR (THE_CHAR='º')  THEN THE_CHAR:='O'
ELSE

IF (THE_CHAR='ÿ') THEN THE_CHAR:='Y' ELSE

IF (THE_CHAR='ø') OR (THE_CHAR='Ø') THEN THE_CHAR:='0' ELSE

IF (THE_CHAR='×') THEN THE_CHAR:='X' ELSE
IF (THE_CHAR='ƒ') THEN THE_CHAR:='F' ELSE
IF (THE_CHAR='¿') THEN THE_CHAR:='?' ELSE
IF (THE_CHAR='©') OR (THE_CHAR='¢') THEN THE_CHAR:='C' ELSE
IF (THE_CHAR='Ð') THEN THE_CHAR:='D' ELSE

IF (THE_CHAR='ñ') OR (THE_CHAR='Ñ') THEN THE_CHAR:='N';

  IF THE_CHAR='[' THEN THE_CHAR:='(' ELSE
  IF THE_CHAR=']' THEN THE_CHAR:=')';

  IF                    (THE_CHAR=CHR(13))                    AND
(WITH_TRANSFORMATION_OF_CR_IN_BRACKET=TRUE) THEN THE_CHAR:=']';

IF ((THE_CHAR>=' ') AND (THE_CHAR<='Z')) OR (THE_CHAR='_') OR (THE_CHAR=']')
THEN BEGIN {OK} END ELSE
IF (THE_CHAR>=CHR(32)) AND (THE_CHAR<>CHR(127)) AND (THE_CHAR<>CHR(255))
THEN THE_CHAR:='_';
```

APPENDIX B

DELPHI 6 procedures and functions called by the main text

```
************************************Beginning of UNIT RS_EM for TX EM ****************************************************
unit RS_EM;
{APPTYPE CONSOLE}  {avec un $ devant...pour pouvoir afficher une fenêtre
                DOS en cas d'écriture de type "WRITELN('....');"}
INTERFACE
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Buttons,MMSystem;

CONST DEGRE_NOMINAL_RS_EM=63;{si M=6 bits DEGRE_NOMINAL=N=2 puissance 6 -1}
CONST DEGRE_NOMINAL_MOINS_1_RS_EM=62;{63-1}
TYPE POLYNOME_RS=ARRAY[0..DEGRE_NOMINAL_MOINS_1_RS_EM] OF BYTE;
VAR C_X_PRIM_RS_EM,V_X_RS_EM,W_X_RS_EM,L_X_RS_EM,U_X_RS_EM,S_X_RS_EM,D_X_RS_EM,G_X_RS_EM,
C_X_RS_EM,I_X_RS_EM:POLYNOME_RS;
VAR D_L_X_RS_EM,D_S_X_RS_EM:BYTE;

CONST DEUX_FOIS_T_RS_EM=16;{2*T avec T le nombre de symboles pouvant être corrigées (longueur de contrôle)}
{message de 47 symboles, symbole entre 0 et 63, 47 vaut pour X46}
{47=N-2t=63-16}
CONST DEGRE_MESSAGE_RS_EM=DEGRE_NOMINAL_RS_EM-DEUX_FOIS_T_RS_EM;{47 nombre de symboles par message}
CONST DEGRE_MESSAGE_MOINS_1_RS_EM=DEGRE_MESSAGE_RS_EM-1;{46}
TYPE MESSAGE_47_SYMBOLES_RS_EM=ARRAY[0..DEGRE_MESSAGE_MOINS_1_RS_EM] OF BYTE;
VAR
MESSAGE_ENVOYE_RS_EM:MESSAGE_47_SYMBOLES_RS_EM=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38
,39,40,41,42,43,44,45,46,47);

PROCEDURE DIVISION_I_X_X_2_T_PAR_G_X_EM(MESSAGE_I_X:MESSAGE_47_SYMBOLES_RS_EM);
PROCEDURE DETERMINATION_DE_C_X_EM;
PROCEDURE DETERMINATION_DU_SYNDROME_POLYNOMIAL_EM;
PROCEDURE ALGORITHME_D_EUCLIDE_POUR_LES_POLYNOMES_EM;
PROCEDURE CALCUL_DES_POSITIONS_DES_ERREURS_EM(VAR TROP_D_ERREURS:BOOLEAN;VAR NOMBRE_D_ERREURS:BYTE);
PROCEDURE CALCUL_DES_DERIVEES_DE_L_X_EM;
PROCEDURE CALCUL_DE_LA_VALEUR_DES_ERREURS_EM;
PROCEDURE CALCUL_POLYNOME_DE_CORRECTION_PUIS_CORRECTION_EM;
FUNCTION MULTIPLICATION_CORPS_DE_GALOIS_EM(NOMBRE1:BYTE;NOMBRE2:BYTE):BYTE;
PROCEDURE INITIALISATION_RS_DECISION_DURE_EM_AND_GENERATE_GF_AND_DETERMINATION_DE_G_X_EM;

IMPLEMENTATION
uses FONCTIONS_PASCAL,TRADUCTEUR_EDITEUR;


VAR V_E,D_L_X,P,R_L_X:ARRAY[1..DEUX_FOIS_T_RS_EM] OF BYTE;
```

```
VAR N_R:BYTE;
VAR W_X_A_0:BOOLEAN;

VAR NOMBRE1,NOMBRE2:POLYNOME_RS;
VAR TABLEAU_DES_63_POLYNOMES:ARRAY[0..DEGRE_NOMINAL_MOINS_1_RS_EM] OF POLYNOME_RS;

TYPE TABLEAU_DE_63_LONGINT_EM_BASE0=ARRAY[0..DEGRE_NOMINAL_RS_EM] OF LONGINT;
VAR A_TO,I_OF:TABLEAU_DE_63_LONGINT_EM_BASE0;
VAR _P: ARRAY[0..6] OF LONGINT;
VAR INIT_ZERO_EM:LONGINT;{possible choice for g(x): Pi((X+alpha^i)), for i = init_zero, init_zero + 1, ..., init_zero+length-k-1}

{**********}

FUNCTION CONSTRUCTION_POLYNOME_I_X(LE_MESSAGE:MESSAGE_47_SYMBOLES_RS_EM):POLYNOME_RS;
VAR P_I:POLYNOME_RS;
VAR COMPTEUR:BYTE;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO P_I[COMPTEUR]:=0;
 FOR COMPTEUR:=0 TO DEGRE_MESSAGE_MOINS_1_RS_EM DO P_I[COMPTEUR]:=LE_MESSAGE[COMPTEUR];
 CONSTRUCTION_POLYNOME_I_X:=P_I;
END;

{**********}

FUNCTION DECALAGE_POLYNOME_MULTIPIE_PAR_XN(P_I:POLYNOME_RS;POSITIONS:BYTE):POLYNOME_RS;
VAR COMPTEUR,COMPTEUR2:BYTE;
BEGIN
 FOR COMPTEUR:=1 TO POSITIONS DO
 BEGIN
  FOR COMPTEUR2:=DEGRE_NOMINAL_MOINS_1_RS_EM DOWNTO 1 DO
  BEGIN
   P_I[COMPTEUR2]:=P_I[COMPTEUR2-1];
  END;
  P_I[0]:=0;
 END;
 DECALAGE_POLYNOME_MULTIPIE_PAR_XN:=P_I;
END;

{**********}

FUNCTION ADDITION_POLYNOMES_QUELCONQUES(P1:POLYNOME_RS;P2:POLYNOME_RS):POLYNOME_RS;
VAR COMPTEUR:BYTE;
VAR P_I:POLYNOME_RS;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  P_I[COMPTEUR]:=(P1[COMPTEUR]) XOR (P2[COMPTEUR]);
 END;
 ADDITION_POLYNOMES_QUELCONQUES:=P_I;
END;
```

```
{**********}

FUNCTION MULTIPLICATION_POLYNOME_PAR_CONSTANTE(P1:POLYNOME_RS;C:BYTE):POLYNOME_RS;
VAR COMPTEUR:BYTE;
VAR P_I:POLYNOME_RS;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  P_I[COMPTEUR]:=MULTIPLICATION_CORPS_DE_GALOIS_EM(P1[COMPTEUR],C);
 END;
 MULTIPLICATION_POLYNOME_PAR_CONSTANTE:=P_I;
END;

{**********}

FUNCTION EXTRACTION_DEGRE_MAX_DU_POLYNOME(P1:POLYNOME_RS):BYTE;
VAR DP:BYTE;
VAR COMPTEUR:BYTE;
BEGIN
 DP:=DEGRE_NOMINAL_MOINS_1_RS_EM;
 FOR COMPTEUR:=DEGRE_NOMINAL_MOINS_1_RS_EM DOWNTO 0 DO
 BEGIN
  IF P1[COMPTEUR]<>0 THEN BREAK;
  IF DP=0 THEN DP:=255 ELSE DEC(DP);
 END;
 EXTRACTION_DEGRE_MAX_DU_POLYNOME:=DP;
END;

{**********}

FUNCTION DIVISION_DE_2_NOMBRES_DANS_GF2m(N,D:BYTE):BYTE;
VAR COMPTEUR:BYTE;
VAR Q,R:BYTE;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_RS_EM DO
 BEGIN
  Q:=COMPTEUR;
  R:=MULTIPLICATION_CORPS_DE_GALOIS_EM(Q,D);
  R:=N XOR R;
  IF R=0 THEN BREAK;
 END;
 DIVISION_DE_2_NOMBRES_DANS_GF2m:=Q;
 { WRITELN(Q);}
END;

{**********}

PROCEDURE DIVISION_POLYNOMES_QUELCONQUES(P1:POLYNOME_RS;P2:POLYNOME_RS);
VAR COMPTEUR,COMPTEUR2:BYTE;
VAR Q_FOIS_D:POLYNOME_RS;
VAR DP1,DP2:BYTE;
```

```
VAR DQ:BYTE;
VAR CQ:BYTE;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO NOMBRE2[COMPTEUR]:=P1[COMPTEUR];
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO NOMBRE1[COMPTEUR]:=0;

 NOMBRE2:=P1;
 DP1:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(P1);
 DP2:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(P2);
 IF DP1=255 THEN EXIT;
 IF DP2=255 THEN BEGIN EXIT; END;

 WHILE (DP1>=DP2) AND (DP1<>255) DO
 BEGIN
  DQ:=DP1-DP2;
  CQ:=DIVISION_DE_2_NOMBRES_DANS_GF2m(NOMBRE2[DP1],P2[DP2]);
  NOMBRE1[DQ]:=CQ;

  Q_FOIS_D:=MULTIPLICATION_POLYNOME_PAR_CONSTANTE(P2,CQ);
  Q_FOIS_D:=DECALAGE_POLYNOME_MULTIPIE_PAR_XN(Q_FOIS_D,DQ);
  NOMBRE2:=ADDITION_POLYNOMES_QUELCONQUES(NOMBRE2,Q_FOIS_D);
  DP1:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(NOMBRE2);
 END;
END;

{**********}

FUNCTION MULTIPLICATION_POLYNOMES_QUELCONQUES(P1:POLYNOME_RS;P2:POLYNOME_RS):POLYNOME_RS;
VAR COMPTEUR,COMPTEUR2:BYTE;
VAR P_I:POLYNOME_RS;
VAR DP2:BYTE;
BEGIN
 DP2:=DEGRE_NOMINAL_MOINS_1_RS_EM;
 FOR COMPTEUR:=DEGRE_NOMINAL_MOINS_1_RS_EM DOWNTO 0 DO
 BEGIN
  IF P2[COMPTEUR]<>0 THEN BREAK;
  IF DP2>0 THEN DEC(DP2);
 END;

 FOR COMPTEUR:=0 TO DP2 DO
 BEGIN
  FOR COMPTEUR2:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO TABLEAU_DES_63_POLYNOMES[COMPTEUR][COMPTEUR2]:=0;
  P_I:=DECALAGE_POLYNOME_MULTIPIE_PAR_XN(P1,COMPTEUR);
  P_I:=MULTIPLICATION_POLYNOME_PAR_CONSTANTE(P_I,P2[COMPTEUR]);
  TABLEAU_DES_63_POLYNOMES[COMPTEUR]:=P_I;
 END;

 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO P_I[COMPTEUR]:=0;

 FOR COMPTEUR:=0 TO DP2 DO
 BEGIN
```

```
 P_I:=ADDITION_POLYNOMES_QUELCONQUES(P_I,TABLEAU_DES_63_POLYNOMES[COMPTEUR]);
 END;
 MULTIPLICATION_POLYNOMES_QUELCONQUES:=P_I;
END;


{**********}

PROCEDURE DIVISION_I_X_X_2_T_PAR_G_X_EM(MESSAGE_I_X:MESSAGE_47_SYMBOLES_RS_EM);
VAR I_X_X_2_T:POLYNOME_RS;
VAR COMPTEUR:BYTE;
BEGIN
 I_X_RS_EM:=CONSTRUCTION_POLYNOME_I_X(MESSAGE_I_X);
 I_X_X_2_T:=DECALAGE_POLYNOME_MULTIPIE_PAR_XN(I_X_RS_EM,DEUX_FOIS_T_RS_EM);

 DIVISION_POLYNOMES_QUELCONQUES(I_X_X_2_T,G_X_RS_EM);
END;

{**********}

PROCEDURE DETERMINATION_DE_C_X_EM;
VAR COMPTEUR:BYTE;
BEGIN
 C_X_RS_EM:=MULTIPLICATION_POLYNOMES_QUELCONQUES(NOMBRE1,G_X_RS_EM);
END;

{**********}

PROCEDURE DETERMINATION_DU_SYNDROME_POLYNOMIAL_EM;
TYPE TABLEAU_DE_0_A_N_1=ARRAY[0..DEGRE_NOMINAL_MOINS_1_RS_EM] OF BYTE;
TYPE TABLEAU_DE_0_A_2T_1=ARRAY[0..DEUX_FOIS_T_RS_EM-1] OF BYTE;
VAR AI:TABLEAU_DE_0_A_2T_1;
VAR D_AI:TABLEAU_DE_0_A_2T_1;
VAR X:TABLEAU_DE_0_A_N_1;
VAR E_D_AI:TABLEAU_DE_0_A_N_1;
VAR X1,XJ,I,J,P,M:BYTE;
BEGIN
 FOR I:=0 TO DEUX_FOIS_T_RS_EM-1 DO
 BEGIN
  P:=INIT_ZERO_EM+I;
  AI[I]:=A_TO[P];
 END;

 FOR I:=0 TO DEUX_FOIS_T_RS_EM-1 DO
 BEGIN
  D_AI[I]:=0;
  X[0]:=1; X[1]:=AI[I]; X1:=X[1];XJ:=X1;
  FOR J:=2 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
  BEGIN
   XJ:=MULTIPLICATION_CORPS_DE_GALOIS_EM(XJ,X1);
   X[J]:=XJ;
```

```
 END;

 FOR J:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  E_D_Ai[J]:=MULTIPLICATION_CORPS_DE_GALOIS_EM(D_X_RS_EM[J],X[J]);
  D_AI[I]:=D_AI[I] XOR E_D_Ai[J];{scalaire}
 END;
END;

FOR I:=0 TO DEUX_FOIS_T_RS_EM-1 DO
BEGIN
 S_X_RS_EM[I]:=D_AI[I];
END;

D_S_X_RS_EM:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(S_X_RS_EM);
END;

{**********}

PROCEDURE ALGORITHME_D_EUCLIDE_POUR_LES_POLYNOMES_EM;
VAR TOURS,COMPTEUR,D_R_I:LONGINT;
VAR a,b:POLYNOME_RS;
VAR r_I_M_1,r_I_P_1,r_I:POLYNOME_RS;
VAR u_I_M_1,u_I_P_1,u_I:POLYNOME_RS;
VAR v_I_M_1,v_I_P_1,v_I:POLYNOME_RS;
VAR q_I_P_1:POLYNOME_RS;
LABEL POINT_2;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO a[COMPTEUR]:=0;
 a[DEUX_FOIS_T_RS_EM]:=1;

 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  u_I_M_1[COMPTEUR]:=0;
  v_I_M_1[COMPTEUR]:=0;
  u_I[COMPTEUR]:=0;
  v_I[COMPTEUR]:=0;
 END;
 u_I_M_1[0]:=1;
 v_I[0]:=1;
 b:=S_X_RS_EM;
 r_I_M_1:=a;
 r_I:=b;

 TOURS:=0;
 POINT_2:{**********************}
 INC(TOURS);
 IF TOURS>=100 THEN
 BEGIN
  D_L_X_RS_EM:=255;
  EXIT;
```

```
 END;
 DIVISION_POLYNOMES_QUELCONQUES(r_I_M_1,r_I);
 q_I_P_1:=NOMBRE1;
 r_I_P_1:=MULTIPLICATION_POLYNOMES_QUELCONQUES(q_I_P_1,r_I);
 r_I_P_1:=ADDITION_POLYNOMES_QUELCONQUES(r_I_M_1,r_I_P_1);
 u_I_P_1:=MULTIPLICATION_POLYNOMES_QUELCONQUES(q_I_P_1,u_I);
 u_I_P_1:=ADDITION_POLYNOMES_QUELCONQUES(u_I_M_1,u_I_P_1);
 v_I_P_1:=MULTIPLICATION_POLYNOMES_QUELCONQUES(q_I_P_1,v_I);
 v_I_P_1:=ADDITION_POLYNOMES_QUELCONQUES(v_I_M_1,v_I_P_1);
 D_R_I:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(r_I);
 IF D_R_I>(DEUX_FOIS_T_RS_EM DIV 2)-1 THEN
 BEGIN
  r_I_M_1:=r_I; r_I:=r_I_P_1;
  u_I_M_1:=u_I; u_I:=u_I_P_1;
  v_I_M_1:=v_I; v_I:=v_I_P_1;

  GOTO POINT_2;
 END;
 L_X_RS_EM:=v_I;
 W_X_RS_EM:=r_I;
 U_X_RS_EM:=u_I;
 D_L_X_RS_EM:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(L_X_RS_EM);

 W_X_A_0:=TRUE;
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  IF W_X_RS_EM[COMPTEUR]<>0 THEN BEGIN W_X_A_0:=FALSE; BREAK; END;
 END;
END;

{**********}

FUNCTION CALCUL_POLYNOME_EN_FONCTION_DE_X(P_I:POLYNOME_RS;X:BYTE):BYTE;
VAR S:BYTE;
VAR XX,F:BYTE;
VAR COMPTEUR,COMPTEUR2:BYTE;
BEGIN
 S:=0;
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  F:=P_I[COMPTEUR];
  IF F>0 THEN
  BEGIN
   XX:=1;
   FOR COMPTEUR2:=1 TO COMPTEUR DO XX:=MULTIPLICATION_CORPS_DE_GALOIS_EM(XX,X);
   F:=MULTIPLICATION_CORPS_DE_GALOIS_EM(F,XX);
  END;
  S:=S XOR F;
 END;
 CALCUL_POLYNOME_EN_FONCTION_DE_X:=S;
END;
```

```
{**********}

FUNCTION CALCUL_DERIVEE_DE_L_X_EM_AU_POINT(B:BYTE):BYTE;
VAR K,AK,BK,BK_PLUS_1,CK,CK_PLUS_1:BYTE;
LABEL POINT_2;
VAR COMPTEUR:BYTE;
BEGIN
 K:=D_L_X_RS_EM;
 BK:=L_X_RS_EM[K];
 AK:=BK;
 BK_PLUS_1:=0;
 CK_PLUS_1:=0;

 POINT_2:
 BK:=MULTIPLICATION_CORPS_DE_GALOIS_EM(B,BK_PLUS_1);
 BK:=AK XOR BK;

 IF K>0 THEN
 BEGIN
  CK:=MULTIPLICATION_CORPS_DE_GALOIS_EM(B,CK_PLUS_1);
  CK:=BK XOR CK;
 END;

 IF K>0 THEN
 BEGIN
  DEC(K);
  CK_PLUS_1:=CK;
  BK_PLUS_1:=BK;
  AK:=L_X_RS_EM[K];
  GOTO POINT_2;
 END;
 CALCUL_DERIVEE_DE_L_X_EM_AU_POINT:=CK;
END;

{**********}

PROCEDURE CALCUL_DES_DERIVEES_DE_L_X_EM;
VAR I:BYTE;
VAR COMPTEUR:BYTE;
BEGIN
 FOR COMPTEUR:=1 TO N_R DO
 BEGIN
  D_L_X[COMPTEUR]:=CALCUL_DERIVEE_DE_L_X_EM_AU_POINT(R_L_X[COMPTEUR]);
 END;
END;

{**********}

PROCEDURE CALCUL_DE_LA_VALEUR_DES_ERREURS_EM;
VAR I:BYTE;
```

```
VAR COMPTEUR,COMPTEUR2:BYTE;
VAR W_B,Xj_1_b,Xj_1:BYTE;
BEGIN
 FOR COMPTEUR:=1 TO N_R DO
 BEGIN
  Xj_1:=R_L_X[COMPTEUR];
  Xj_1_b:=1; FOR COMPTEUR2:=1 TO 3 DO Xj_1_b:=MULTIPLICATION_CORPS_DE_GALOIS_EM(Xj_1_b,Xj_1);
  W_B:=CALCUL_POLYNOME_EN_FONCTION_DE_X(W_X_RS_EM,Xj_1);
  W_B:=MULTIPLICATION_CORPS_DE_GALOIS_EM(W_B,Xj_1_b);
  V_E[COMPTEUR]:=DIVISION_DE_2_NOMBRES_DANS_GF2m(W_B,D_L_X[COMPTEUR]);
 END;
END;

{**********}

PROCEDURE CALCUL_POLYNOME_DE_CORRECTION_PUIS_CORRECTION_EM;
VAR COMPTEUR:BYTE;
VAR R:BOOLEAN;
BEGIN
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO V_X_RS_EM[COMPTEUR]:=0;
 FOR COMPTEUR:=1 TO N_R DO
 BEGIN
  V_X_RS_EM[P[COMPTEUR]]:=V_E[COMPTEUR];
 END;

 C_X_PRIM_RS_EM:=ADDITION_POLYNOMES_QUELCONQUES(D_X_RS_EM,V_X_RS_EM);

 R:=TRUE;
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  IF C_X_RS_EM[COMPTEUR]<>C_X_PRIM_RS_EM[COMPTEUR] THEN BEGIN R:=FALSE; BREAK; END;
 END;
END;

{**********}

PROCEDURE CALCUL_DES_RACINES_DE_L_X_EM_ET_DETERMINATION_DES_POSITIONS_D_ERREURS;
VAR I:BYTE;
VAR COMPTEUR:BYTE;
BEGIN
 FOR COMPTEUR:=1 TO N_R DO
 BEGIN
  I:=I_OF[R_L_X[COMPTEUR]];
  IF I=0 THEN P[COMPTEUR]:=0 ELSE P[COMPTEUR]:=DEGRE_NOMINAL_RS_EM-I;
 END;
END;

{**********}

PROCEDURE CALCUL_DES_POSITIONS_DES_ERREURS_EM(VAR TROP_D_ERREURS:BOOLEAN;VAR NOMBRE_D_ERREURS:BYTE);
VAR COMPTEUR:BYTE;
```

```
VAR L_DE_X:BYTE;
BEGIN
 D_L_X_RS_EM:=EXTRACTION_DEGRE_MAX_DU_POLYNOME(L_X_RS_EM);
 IF D_L_X_RS_EM=255 THEN BEGIN TROP_D_ERREURS:=TRUE; EXIT; END;
 N_R:=0;
 FOR COMPTEUR:=0 TO DEGRE_NOMINAL_RS_EM DO
 BEGIN
  L_DE_X:=CALCUL_POLYNOME_EN_FONCTION_DE_X(L_X_RS_EM,COMPTEUR);
  IF L_DE_X=0 THEN
  BEGIN
   INC(N_R);
   R_L_X[N_R]:=COMPTEUR;
  END;
 END;
 NOMBRE_D_ERREURS:=N_R;
 IF N_R<D_L_X_RS_EM THEN BEGIN TROP_D_ERREURS:=TRUE; EXIT; END
 ELSE TROP_D_ERREURS:=FALSE;
 CALCUL_DES_RACINES_DE_L_X_EM_ET_DETERMINATION_DES_POSITIONS_D_ERREURS;
END;

{***********}

FUNCTION MULTIPLICATION_CORPS_DE_GALOIS_EM(NOMBRE1:BYTE;NOMBRE2:BYTE):BYTE;
VAR I,J,K,W:WORD;
BEGIN
 IF (NOMBRE1=0) OR (NOMBRE2=0) THEN W:=0 ELSE
 BEGIN
  I:=I_OF[NOMBRE1];
  J:=I_OF[NOMBRE2];
  K:=(I+J) MOD DEGRE_NOMINAL_RS_EM;
  W:=A_TO[K];
 END;
 MULTIPLICATION_CORPS_DE_GALOIS_EM:=W;
END;

{**********}

PROCEDURE INITIALISATION_RS_DECISION_DURE_EM_AND_GENERATE_GF_AND_DETERMINATION_DE_G_X_EM;
VAR I,J,M:LONGINT;
BEGIN
//INITIALISATION_RS_DECISION_DURE_EM
 FOR I:=1 TO 5 DO P[i] := 0;
 _P[0]:=1; _P[6]:=1; _P[1]:=1;
 INIT_ZERO_EM:=4;

//GENERATE_GF
  M:=1;
  A_TO[6]:= 0;
  FOR I:=0 TO 5 DO
  BEGIN
   A_TO[I] := M;
```

```
  I_OF[A_TO[I]] := I;
  IF _P[I]<>0 THEN A_TO[6] := A_TO[6] XOR M;
  M:=M SHL 1;
 END;

 I_OF[A_TO[6]]:=6;
 M:=M SHR 1;
 FOR I:=7 TO DEGRE_NOMINAL_MOINS_1_RS_EM DO
 BEGIN
  IF A_TO[I-1] >= M THEN
  A_TO[I]:= A_TO[6] XOR ((A_TO[I-1] XOR M) SHL 1) ELSE
  A_TO[I]:= A_TO[I-1] SHL 1;
  I_OF[A_TO[I]]:=I;
 END;
 I_OF[0]:= -1;

//DETERMINATION_DE_G_X_EM
  G_X_RS_EM[0] := A_TO[INIT_ZERO_EM];
  G_X_RS_EM[1] := 1;
  FOR I:=2 TO DEGRE_NOMINAL_RS_EM-DEGRE_MESSAGE_RS_EM DO//V.4.48.2
  BEGIN
   G_X_RS_EM[I] := 1;
   FOR J:=I-1 DOWNTO 1 DO
   BEGIN
    IF G_X_RS_EM[J]<>0 THEN G_X_RS_EM[J] := G_X_RS_EM[J-1] XOR A_TO[(I_OF[G_X_RS_EM[J]]+I+INIT_ZERO_EM-1) MOD DEGRE_NOMINAL_RS_EM]
    ELSE G_X_RS_EM[J] := G_X_RS_EM[J-1];
   END;
   G_X_RS_EM[0] := A_TO[(I_OF[G_X_RS_EM[0]]+I+INIT_ZERO_EM-1) MOD DEGRE_NOMINAL_RS_EM];
  END;
END;


BEGIN

end.
************************************End UNIT RS_EM for TX EM ******************************************************

{***********}

PROCEDURE CODING_OF_THE_EXTENDED_MESSAGE_ID(MESSAGE_ID:STRING;VAR MESSAGE_ID_TX_SYMBOLS:ARRAY_OF_THE_108_SYMBOLS;VAR
TX_FRAME_LENGTH:BYTE);//104 to 108 in V.4.45.6
{MESSAGE_ID is the message to code (1 to 47 characters)
The goal is to make the extented message ID (MESSAGE_ID_TX_SYMBOLS) composed of 13 characters of 4 bits and a shortened Reed Solomon frame (16+1*3 characters
of 6 bits as a minimum (29 symbols of 4 bits) and 63 characters of 6 bits as a maximum (95 symbols of 4 bits). So the total is between 42 and 108 symbols of 4 bits
TX_FRAME_LENGTH in the number of symbols of 4 bits: 42 to 108}
VAR LONG,COUNTER,COUNTER2,FRAME_LENGTH:WORD;
VAR RESULT_STRING,BLOCK_52_BITS,RS_STRING:STRING;{V.4.45.6}
VAR BIT:CHAR;
VAR SYMBOL:BYTE;{0..15}
//pour test
VAR RS_EM_RX_SYMBOLS:ARRAY_OF_THE_95_SYMBOLS;
```

```
VAR RS,DE:BYTE;
VAR CHAINE_CARACTERES_DU_MESSAGE:STRING;
VAR EM:EM_DETECTION_CODE;{ARRAY[0..12] OF BYTE;}
VAR SUCCESS:BOOLEAN;
VAR FAILURE,RS_SEQUENCE_LENGTH:BYTE;
BEGIN
{ WRITELN('Debut de CODING_OF_THE_EXTENDED_CALL_ID ',MESSAGE_ID);}

 {First the 13 Length frame to say "Extended call ID" with its length}
 FRAME_LENGTH:=LENGTH(MESSAGE_ID);{1 to 47}
 LONG:=FRAME_LENGTH DIV 3; IF FRAME_LENGTH MOD 3<>0 THEN INC(LONG);{LONG between 1 to 16 which will be transformed into "0 to 15"}
 MAKING_OF_THE_DETECTION_FRAME(LONG,BLOCK_52_BITS);
 RESULT_STRING:=BLOCK_52_BITS;
 {WRITELN('L1 ',FRAME_LENGTH,' ',LONG);}

 {initialize RS encoder, if not done until now, rajouter IF DEJA_FAIT_UNE_FOIS=TRUE...}
 IF INITIALISATION_RS_FAITE_POUR_L_EXTENDED_MESSAGE=FALSE THEN
 BEGIN
  INITIALISATION_RS_FAITE_POUR_L_EXTENDED_MESSAGE:=TRUE;
  INITIALISATION_RS_DECISION_DURE_EM_AND_GENERATE_GF_AND_DETERMINATION_DE_G_X_EM;
 END;
 {WRITELN('Mes. ',MESSAGE_ID,' ',Caracteres_de_l_EM_en_majuscules,' ',FRAME_LENGTH,' ',LONG);}
 {Calculation of the 16 parity characters}
 {complete the block of 3 characters with "^"}
 FOR COUNTER:=FRAME_LENGTH+1 TO LONG*3 DO MESSAGE_ID:=MESSAGE_ID+'^';{to fill the remaining room of the block}
 {The scrambling is compulsory for messages having the same character. Done with the number of the character (1 to 47) in the string (included filling '^').
 Without scrambling, there can be several successes with different messages, deleted the 23/3/22 because gives problems}
{ FOR COUNTER:=1 TO LONG*3 DO MESSAGE_ID[COUNTER]:=CHR( ((ORD(MESSAGE_ID[COUNTER])-32) XOR COUNTER)+32);}{0 à 63}

 {complete the message up to 47 symbols with 0 (hidden characters for the parity calculation}
 FOR COUNTER:=LENGTH(MESSAGE_ID)+1 TO 47 DO MESSAGE_ID:=MESSAGE_ID+' ';{' ' i.e CHR(32) give CHR(0) hidden characters}
 {WRITELN('Mes2. ',MESSAGE_ID,'-',LENGTH(MESSAGE_ID),'-');}

// MESSAGE_ID:='12345678901234567890123456789012345678901234567'; {47 caractères de 6 bits pour test}
 {transfer of the string to an array of BYTE (0 à 63)}
 FOR COUNTER:=0 TO 46 DO MESSAGE_ENVOYE_RS_EM[COUNTER]:=ORD(MESSAGE_ID[COUNTER+1])-32;{0 à 63}
 DIVISION_I_X_X_2_T_PAR_G_X_EM(MESSAGE_ENVOYE_RS_EM);
 {il s'agit d'un code systématique, à savoir que le code de parité p(x) (16 characters) se trouve en premier  (c(x)=i(x)*x^2t+p(x) avec le LSB à droite (on  lit de droite à gauche),  cf
"Index" page 5 et 6)}
 DETERMINATION_DE_C_X_EM;
 {Storage in RS_STRING, then transformation in bits}
 RS_STRING:='';
 {nombre de caractères de 6 bits utilisés dans la séquence RS}
 LONG:=LONG*3; IF LONG>47 THEN LONG:=47;
 FOR COUNTER:=0 TO 15+LONG DO RS_STRING:=RS_STRING+CHR(C_X_RS_EM[COUNTER]);
 {WRITELN('L2 ',LENGTH(RS_STRING)); WRITELN ('Sequence TX ');
 FOR COUNTER:=0 TO 62 DO WRITE(C_X_RS_EM[COUNTER],' '); WRITELN;}
 RS_STRING:=TRANSFORMATION_CHAINE_CARACTERES_6_BITS_EN_CHAINE_BINAIRE(RS_STRING);
 RESULT_STRING:=RESULT_STRING+RS_STRING;{52+(19*6 to 63*6) bits, so 166 to 430}
 {nombre de caractères de 6 bits utilisés, 52 bits correspond aux 13 LENGTHS, LONG entre 19 et 47 et 16 les 16 caractères de parité}
 FRAME_LENGTH:=52+(16+LONG)*6;{global number of bits, it could be a multiple of 4 or not.If not it must be added 2 bits "0"}
```

```
IF FRAME_LENGTH MOD 4=2 THEN
BEGIN
 RESULT_STRING:=RESULT_STRING+'00';
 INC(FRAME_LENGTH,2);{so FRAME_LENGTH between 166+2 and 430+2 bits}
END;
{WRITELN('FRAME_LENGTH ',FRAME_LENGTH);}
{Making of the MESSAGE_ID_TX_SYMBOLS from the binary string (1 symbol for 4 bits)}
COUNTER2:=0;SYMBOL:=0;

{RS:=0; DE:=0;}//test
FOR COUNTER:=1 TO FRAME_LENGTH DO {FRAME_LENGTH between 166+2 and 430+2 bits so 42 to 108 symbols}
BEGIN
 INC(COUNTER2);
 BIT:=RESULT_STRING[COUNTER];
 IF (COUNTER2=1) AND (BIT='1') THEN INC(SYMBOL,8) ELSE
 IF (COUNTER2=2) AND (BIT='1') THEN INC(SYMBOL,4) ELSE
 IF (COUNTER2=3) AND (BIT='1') THEN INC(SYMBOL,2) ELSE
 IF (COUNTER2=4) AND (BIT='1') THEN INC(SYMBOL,1);{SYMBOL: 0 to 15}
 IF COUNTER2=4 THEN
 BEGIN
  MESSAGE_ID_TX_SYMBOLS[(COUNTER DIV 4)-1]:=SYMBOL;{(COUNTER DIV 4)-1: 0 to 103 max}
  //test
  {IF COUNTER>9*4 THEN //RS sequence
  BEGIN
   WRITE(SYMBOL,' ');
   RS_EM_RX_SYMBOLS[RS]:=SYMBOL;
   INC(RS);
  END ELSE
  BEGIN //detection code
   WRITE(SYMBOL,' ');
   EM[DE]:=SYMBOL;//0 à 8
   INC(DE);
  END;}
  COUNTER2:=0;SYMBOL:=0;
 END;
END;

 TX_FRAME_LENGTH:=FRAME_LENGTH DIV 4;{42 to 108 symboles de 4 bits}
 {WRITELN('Fin de CODING_OF_THE_EXTENDED_CALL_ID ',LONG,' ',FRAME_LENGTH,' ',TX_FRAME_LENGTH);}
END;

{*********}

FUNCTION POWER(X:INT64;Y:BYTE):INT64;
{X power Y}
VAR COUNTER:BYTE;
VAR THE_RESULT:INT64;
BEGIN
 THE_RESULT:=1;
 IF Y=0 THEN THE_RESULT:=1 ELSE
 BEGIN
```

```
  FOR COUNTER:=1 TO Y DO THE_RESULT:=THE_RESULT*X;
 END;
 POWER:=THE_RESULT;
END;

{**********}

FUNCTION ORDER_CHARACTER_EM_STRING(C:CHAR):BYTE;{0 to 60 for 61 EM possible characters}
VAR NUMBER:BYTE;
BEGIN
 IF C='_' THEN C:='[';
 ORDER_CHARACTER_EM_STRING:=ORD(C)-32;{0 for ' ' to  60 for "\"}
END;

{**********}

PROCEDURE ADDING_FINAL_SPACES_TO_REACH_THE_RIGHT_LENGTH(RIGHT_LENGTH:BYTE;VAR THE_STRING:STRING);
VAR COUNTER,THE_LENGTH:BYTE;
BEGIN
 THE_LENGTH:=LENGTH(THE_STRING);
 FOR COUNTER:=THE_LENGTH+1 TO RIGHT_LENGTH DO THE_STRING:=THE_STRING+' ';
END;

{**********}

PROCEDURE REMOVAL_OF_FINAL_SPACES(VAR CHAINE:STRING);
{From CHAINE, final spaces are removed}
VAR TAMPON:STRING;
VAR COMPTEUR:LONGINT;
VAR CARACTERE:CHAR;
VAR NOMBRE:LONGINT;
BEGIN
 NOMBRE:=0;{initialization}
 FOR COMPTEUR:=LENGTH(CHAINE) DOWNTO 1 DO
 BEGIN
  CARACTERE:=CHAINE[COMPTEUR];
  IF CARACTERE=' ' THEN INC(NOMBRE)
  ELSE BREAK;
 END;
 TAMPON:='';{initialization}
 FOR COMPTEUR:=1 TO LENGTH(CHAINE)-NOMBRE DO
 BEGIN
  CARACTERE:=CHAINE[COMPTEUR];
  TAMPON:=TAMPON+CARACTERE;
 END;
 CHAINE:=TAMPON;
END;

{**********}

PROCEDURE REMOVAL_OF_INITIAL_SPACES(VAR CHAINE:STRING);
```

```
{From CHAINE, initial spaces are removed}
VAR COMPTEUR:LONGINT;
VAR CARACTERE:CHAR;
VAR NOMBRE:LONGINT;
BEGIN
 NOMBRE:=0;{initialization}
 FOR COMPTEUR:=1 TO LENGTH(CHAINE) DO
 BEGIN
  CARACTERE:=CHAINE[COMPTEUR];
  IF CARACTERE=' ' THEN INC(NOMBRE)
  ELSE BREAK;
 END;
 DELETE(CHAINE,1,NOMBRE);
END;

{*************}

PROCEDURE EXTENSION_TO_N_CHARACTERS_WITH_FINAL_SPACES(N:BYTE;VAR CHAINE:STRING);
{From CHAINE, final spaces are added up to N characters}
VAR COMPTEUR:BYTE;
VAR LENGTH_STRING:BYTE;
BEGIN
 LENGTH_STRING:=LENGTH(CHAINE);
 FOR COMPTEUR:=LENGTH_STRING+1 TO N DO CHAINE:=CHAINE+' ';
END;

{*************}

PROCEDURE EXTENSION_TO_N_CHARACTERS_WITH_INITIAL_SPACES(N:BYTE;VAR CHAINE:STRING);
{From CHAINE, final spaces are added up to N characters}
VAR COMPTEUR:BYTE;
VAR LENGTH_STRING:BYTE;
BEGIN
 LENGTH_STRING:=LENGTH(CHAINE);
 FOR COMPTEUR:=LENGTH_STRING+1 TO N DO CHAINE:=' '+CHAINE;
END;

{**********}

FUNCTION TRANSFORMATION_N_FINAL_IN_CALL_SIGN_STRING(N_FINAL:INT64;NUMBER_OF_CHARACTERS_OF_THE_CALL_SIGN_STRING:BYTE):STRING;
{38 characters are available for a calls sign: A..B 0..9 "/" and space}
VAR CALL_SIGN_STRING:STRING;
VAR COUNTER:BYTE;
VAR QUOTIENT,NUMBER,POWER_OF_38:INT64;
VAR CALL_SIGN_CHARACTER:CHAR;

 {!!!!!!!!!!!}
 FUNCTION TRANSFORMATION_OF_QUOTIENT_IN_CALL_SIGN_CHARACTER(QUOTIENT:INT64):CHAR;
 VAR C:CHAR;
 BEGIN
  IF QUOTIENT=0 THEN C:=' ' ELSE
```

```
 IF QUOTIENT=37 THEN C:='/' ELSE
 IF (QUOTIENT>=1) AND (QUOTIENT<=26) THEN C:=CHR(QUOTIENT+64) ELSE{letters}
 C:=CHR(QUOTIENT+21);{figures}
 TRANSFORMATION_OF_QUOTIENT_IN_CALL_SIGN_CHARACTER:=C;
END;

BEGIN
 CALL_SIGN_STRING:='';
 NUMBER:=N_FINAL;
 FOR COUNTER:=(NUMBER_OF_CHARACTERS_OF_THE_CALL_SIGN_STRING-1) DOWNTO 0 DO
 BEGIN
  POWER_OF_38:=POWER(38,COUNTER);
  QUOTIENT:=NUMBER DIV POWER_OF_38;//0 to 37
  CALL_SIGN_CHARACTER:=TRANSFORMATION_OF_QUOTIENT_IN_CALL_SIGN_CHARACTER(QUOTIENT);
  CALL_SIGN_STRING:=CALL_SIGN_STRING+CALL_SIGN_CHARACTER;
  DEC(NUMBER,POWER_OF_38*QUOTIENT);
 END;
 TRANSFORMATION_N_FINAL_IN_CALL_SIGN_STRING:=CALL_SIGN_STRING;
END;

{**********}

FUNCTION TRANSFORMATION_N_FINAL_IN_EM_STRING(N_FINAL:INT64;NUMBER_OF_CHARACTERS_OF_THE_EM_STRING:BYTE):STRING;
{61 characters are available for an EM string: CHR(32) to CHR(95) minus ^ [ and ] (filling, reserved and "CHR(10)+CHR(13"}
VAR EM_STRING:STRING;
VAR COUNTER:BYTE;
VAR QUOTIENT,NUMBER,POWER_OF_61:INT64;
VAR EM_CHARACTER:CHAR;

 {!!!!!!!!!!!}
 FUNCTION TRANSFORMATION_OF_QUOTIENT_IN_EM_CHARACTER(QUOTIENT:INT64):CHAR;
 VAR EM_CHAR:CHAR;
 BEGIN
  EM_CHAR:=CHR(32+QUOTIENT);{32 min to 32+61-1=92}
  IF EM_CHAR='[' THEN EM_CHAR:='_';
  TRANSFORMATION_OF_QUOTIENT_IN_EM_CHARACTER:=EM_CHAR;
 END;

BEGIN
 EM_STRING:='';
 NUMBER:=N_FINAL;
 FOR COUNTER:=(NUMBER_OF_CHARACTERS_OF_THE_EM_STRING-1) DOWNTO 0 DO
 BEGIN
  POWER_OF_61:=POWER(61,COUNTER);
  QUOTIENT:=NUMBER DIV POWER_OF_61;
  EM_CHARACTER:=TRANSFORMATION_OF_QUOTIENT_IN_EM_CHARACTER(QUOTIENT);
  EM_STRING:=EM_STRING+EM_CHARACTER;
  DEC(NUMBER,POWER_OF_61*QUOTIENT);
 END;
 TRANSFORMATION_N_FINAL_IN_EM_STRING:=EM_STRING;
END;
```

```
{***********}

PROCEDURE DECODING_OF_THE_EM_CALL_SIGN_IN_CALL_SIGN(EM_CALL_SIGN:STRING;VAR CALL_SIGN:STRING);
{From the 11 EM characters forming the call sign it is extracted the 12 characters maximum of the call sign}
VAR N_FINAL:INT64;
VAR COUNTER:BYTE;
VAR A:ARRAY[1..11] OF BYTE;
BEGIN
 FOR COUNTER:=1 TO 11 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_CALL_SIGN[COUNTER]);{0 to 60}
 {max N_FINAL=61^11=4.35E19 > 38^12=9.065E18}
 A[1]:=A[1] MOD 13;  {A[1] is between always 0 and 12 because 13x61^10=9.27E18>9.065E18. So 2 more bits could be extracted. They are reserved for the future}

 N_FINAL:=A[1]*POWER(61,10)+A[2]*POWER(61,9)+
   A[3]*POWER(61,8)+A[4]*POWER(61,7)+A[5]*POWER(61,6)+A[6]*POWER(61,5)+
   A[7]*POWER(61,4)+A[8]*POWER(61,3)+A[9]*POWER(61,2)+A[10]*61+A[11];
 CALL_SIGN:=TRANSFORMATION_N_FINAL_IN_CALL_SIGN_STRING(N_FINAL,12);
 {WRITELN('CALL_SIGN ',N_FINAL,' -',CALL_SIGN,'-');}
END;


{***********}

PROCEDURE CODING_OF_THE_FREE_HAM_CALL_SIGN_IN_EM_STRING(CALL_SIGN:STRING;VAR EM_CALL_SIGN:STRING;VAR FAILURE:BYTE);
{Free call signs with 11 characters (A..Z, 0..9 or / plus the space to fill, so 38 possible characters) on 11 EM characters}
VAR COUNTER,THE_LENGTH:BYTE;
VAR THE_CHAR:CHAR;
VAR A:ARRAY[1..12] OF BYTE;
VAR VALUE,N_FINAL:INT64;

 {!!!!!!!!!}
 FUNCTION ORDER_CHARACTER_FREE_CALL_SIGN(C:CHAR):BYTE;
 {transforms space in 0, a letter in 1..26, a figure in 27..36, / in 37}
 VAR NUMBER:BYTE;
 BEGIN
  IF C=' ' THEN NUMBER:=0 ELSE
  IF (C>='A') AND (C<='Z') THEN NUMBER:=ORD(C)-64 ELSE
  IF (C>='0') AND (C<='9') THEN NUMBER:=ORD(C)-21{-21=-48+27} ELSE
  IF C='/' THEN NUMBER:=37;
  ORDER_CHARACTER_FREE_CALL_SIGN:=NUMBER;
 END;

BEGIN
 {initialization}
 FAILURE:=0;
 THE_LENGTH:=LENGTH(CALL_SIGN);

 IF (THE_LENGTH<3) OR (THE_LENGTH>12) THEN BEGIN FAILURE:=1; EXIT; END;

 {verification of the characters}
 FOR COUNTER:=1 TO THE_LENGTH DO
```

```
BEGIN
 THE_CHAR:=CALL_SIGN[COUNTER];

 {all characters in capitals}
 IF THE_CHAR>='a' THEN
 BEGIN
  THE_CHAR:=CHR(ORD(THE_CHAR)-32);
  CALL_SIGN[COUNTER]:=THE_CHAR;
 END;

 {authorized Ham characters: A..Z, 0..9 and "/"}
 IF ((THE_CHAR>='A') AND (THE_CHAR<='Z')) OR ((THE_CHAR>='0') AND (THE_CHAR<='9')) OR (THE_CHAR='/') THEN
 BEGIN END ELSE BEGIN FAILURE:=2; EXIT; END;
END;

{"/" can't be located at the beginning or at the end of the call sign. This exception could be used in the future}
IF (CALL_SIGN[1]='/') OR (CALL_SIGN[THE_LENGTH]='/') THEN BEGIN FAILURE:=3; EXIT; END;

FOR COUNTER:=1 TO THE_LENGTH DO A[COUNTER]:=ORDER_CHARACTER_FREE_CALL_SIGN(CALL_SIGN[COUNTER]);{0 to 36}
FOR COUNTER:=THE_LENGTH+1 TO 12 DO A[COUNTER]:=0;{0 for no character (space)}

{max N_FINAL=38^12=9.065E18 compatible with the maximum of INT64: 9.223E18, for a maximum of 61^11=4.35E19 (4.8 more).
So 2 bits are available. They are reserved for the future}
N_FINAL:=A[1]*POWER(38,11)+A[2]*POWER(38,10)+
  A[3]*POWER(38,9)+A[4]*POWER(38,8)+A[5]*POWER(38,7)+A[6]*POWER(38,6)+
  A[7]*POWER(38,5)+A[8]*POWER(38,4)+A[9]*POWER(38,3)+A[10]*POWER(38,2)+A[11]*POWER(38,1)+A[12];
EM_CALL_SIGN:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_FINAL,11);
{Note EM_CALL_SIGN[1] is between always 0 and 12 because 13x61^10=9.27E18>9.065E18}
{WRITELN('Final ',N_FINAL,' -',EM_CALL_SIGN,'-',ORD(EM_CALL_SIGN[1])-32);}
END;

{**********}

PROCEDURE DECODING_OF_THE_EM_LOCATOR_IN_LOCATOR(EM_LOCATOR:STRING;VAR LOCATOR:STRING);
{From the 5 EM characters forming the Locator it is extracted the 6 characters Locator}
VAR N_L:INT64;
VAR COUNTER:BYTE;
VAR A:ARRAY[1..5] OF BYTE;
VAR CHAR1,CHAR2,CHAR3,CHAR4,CHAR5,CHAR6:CHAR;
 {!!!!!!!}
 FUNCTION LETTER(C:BYTE):CHAR;
 {From a code representing a letter in 0..25, transforms in character}
 BEGIN
  LETTER:=CHR(C+65);
 END;

 {!!!!!!!}
 FUNCTION FIGURE(C:BYTE):CHAR;
 {From a code representing a figure in 0..9 transforms in character}
 BEGIN
  FIGURE:=CHR(C+48);
```

```
 END;

 {!!!!!!!!}
 FUNCTION ORDER_LETTER(C:CHAR):BYTE;
 {transforms a letter in 0..25}
 BEGIN
  ORDER_LETTER:=ORD(C)-65;
 END;

 {!!!!!!!!}
 FUNCTION ORDER_FIGURE(C:CHAR):BYTE;
 {transforms a figure in 0..9}
 BEGIN
  ORDER_FIGURE:=ORD(C)-48;
 END;


BEGIN
 FOR COUNTER:=1 TO 5 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_LOCATOR[COUNTER]);{0 to 60}
 N_L:=A[1]*POWER(61,4)+A[2]*POWER(61,3)+A[3]*POWER(61,2)+A[4]*61+A[5];
 N_L:=N_L MOD 18662400; {to take profit of 5 bits in the the future}

 {determination of the Locator}
 CHAR1:=LETTER(N_L DIV (18*57600)); N_L:=N_L-ORDER_LETTER(CHAR1)*(18*57600);
 CHAR2:=LETTER(N_L DIV 57600); N_L:=N_L-ORDER_LETTER(CHAR2)*57600;
 CHAR3:=FIGURE(N_L DIV 5760); N_L:=N_L-ORDER_FIGURE(CHAR3)*5760;
 CHAR4:=FIGURE(N_L DIV 576); N_L:=N_L-ORDER_FIGURE(CHAR4)*576;
 CHAR5:=LETTER(N_L DIV 24); N_L:=N_L-ORDER_LETTER(CHAR5)*24;
 CHAR6:=LETTER(N_L);
 LOCATOR:=CHAR1+CHAR2+CHAR3+CHAR4+CHAR5+CHAR6;
END;

{**********}

PROCEDURE CHECK_AND_FORMATTING_OF_THE_LOCATOR(VAR LOCATOR:STRING;VAR FAILURE:BYTE);
VAR THE_LENGTH,COUNTER:BYTE;
VAR THE_CHAR:CHAR;
BEGIN
 FAILURE:=0;{initialization}

 {Locator management, must be 4 or 6 characters}
 THE_LENGTH:=LENGTH(LOCATOR);
 IF (THE_LENGTH<4) OR (THE_LENGTH=5) OR (THE_LENGTH>6) THEN
 BEGIN FAILURE:=1; EXIT; END;

 {If 4 characters Locator, it is added LL, au milieu de XX}
 IF THE_LENGTH=4 THEN LOCATOR:=LOCATOR+'LL';

 {all characters in capitals}
 FOR COUNTER:=1 TO 6 DO
 BEGIN
```

```
 THE_CHAR:=LOCATOR[COUNTER];
 IF THE_CHAR>='a' THEN
 BEGIN THE_CHAR:=CHR(ORD(THE_CHAR)-32); LOCATOR[COUNTER]:=THE_CHAR; END;
END;

 {verification of the Locator}
 IF ((LOCATOR[1]<'A') OR (LOCATOR[1]>'R')) OR ((LOCATOR[2]<'A') OR (LOCATOR[2]>'R')) OR
    ((LOCATOR[3]<'0') OR (LOCATOR[3]>'9')) OR ((LOCATOR[4]<'0') OR (LOCATOR[4]>'9')) OR
    ((LOCATOR[5]<'A') OR (LOCATOR[5]>'X')) OR ((LOCATOR[6]<'A') OR (LOCATOR[6]>'X')) THEN
 BEGIN FAILURE:=2; EXIT; END;
END;

{***********}

PROCEDURE CODING_OF_THE_LOCATOR_IN_EM_STRING(LOC:STRING;VAR EM_LOCATOR:STRING);
{Locator of 6 characters  (AA to RR / 00 to 99 / AA to XX, so 18*18*10*10*24*24=18 662 400 combinations) on 5 EM characters (61^5=8.44E8>18 662 400, ratio=45.25,
so 5 more bits are available. They are reserved for the future}
VAR N_FINAL:INT64;
BEGIN
 {calculation of the Locator AA to RR / 00 to 99 / AA to XX}
 {Maximum=18*18*100*24*24-1= 18662399}
 N_FINAL:=(ORD(LOC[1])-65)*18*57600+(ORD(LOC[2])-65)*57600+
   (ORD(LOC[3])-48)*5760+(ORD(LOC[4])-48)*576+(ORD(LOC[5])-65)*24+(ORD(LOC[6])-65);{Maximum 18662399}
 EM_LOCATOR:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_FINAL,5);
END;

{***********}

PROCEDURE DECODING_OF_THE_POWER(EM_POWER:STRING;VAR POWER_STRING:STRING);
VAR N_P:BYTE;
VAR P:STRING;
BEGIN
 N_P:=ORDER_CHARACTER_EM_STRING(EM_POWER[1]);{0 to 60}
 IF N_P=0 THEN P:='1 mW' ELSE IF N_P=1 THEN P:='1.5 mW' ELSE IF N_P=2 THEN P:='2 mW' ELSE IF N_P=3 THEN P:='3 mW' ELSE IF N_P=4 THEN P:='4 mW' ELSE
 IF N_P=5 THEN P:='5 mW' ELSE IF N_P=6 THEN P:='6 mW' ELSE IF N_P=7 THEN P:='7 mW' ELSE IF N_P=8 THEN P:='8 mW' ELSE IF N_P=9 THEN P:='9 mW' ELSE
 IF N_P=10 THEN P:='10 mW' ELSE IF N_P=11 THEN P:='15 mW' ELSE IF N_P=12 THEN P:='20 mW' ELSE IF N_P=13 THEN P:='30 mW' ELSE IF N_P=14 THEN P:='40 mW'
ELSE
 IF N_P=15 THEN P:='50 mW' ELSE IF N_P=16 THEN P:='60 mW' ELSE IF N_P=17 THEN P:='70 mW' ELSE IF N_P=18 THEN P:='80 mW' ELSE IF N_P=19 THEN P:='90 mW'
ELSE
 IF N_P=20 THEN P:='100 mW' ELSE IF N_P=21 THEN P:='150 mW' ELSE IF N_P=22 THEN P:='200 mW' ELSE IF N_P=23 THEN P:='300 mW' ELSE IF N_P=24 THEN P:='400
mW' ELSE
 IF N_P=25 THEN P:='500 mW' ELSE IF N_P=26 THEN P:='600 mW' ELSE IF N_P=27 THEN P:='700 mW' ELSE IF N_P=28 THEN P:='800 mW' ELSE IF N_P=29 THEN P:='900
mW' ELSE
 IF N_P=30 THEN P:='1 W' ELSE IF N_P=31 THEN P:='1.5 W' ELSE IF N_P=32 THEN P:='2 W' ELSE IF N_P=33 THEN P:='3 W' ELSE IF N_P=34 THEN P:='4 W' ELSE
 IF N_P=35 THEN P:='5 W' ELSE IF N_P=36 THEN P:='6 W' ELSE IF N_P=37 THEN P:='7 W' ELSE IF N_P=38 THEN P:='8 W' ELSE IF N_P=39 THEN P:='9 W' ELSE
 IF N_P=40 THEN P:='10 W' ELSE IF N_P=41 THEN P:='15 W' ELSE IF N_P=42 THEN P:='20 W' ELSE IF N_P=43 THEN P:='30 W' ELSE IF N_P=44 THEN P:='40 W' ELSE
 IF N_P=45 THEN P:='50 W' ELSE IF N_P=46 THEN P:='60 W' ELSE IF N_P=47 THEN P:='70 W' ELSE IF N_P=48 THEN P:='80 W' ELSE IF N_P=49 THEN P:='90 W' ELSE
 IF N_P=50 THEN P:='100 W' ELSE IF N_P=51 THEN P:='150 W' ELSE IF N_P=52 THEN P:='200 W' ELSE IF N_P=53 THEN P:='300 W' ELSE IF N_P=54 THEN P:='400 W'
ELSE
 IF N_P=55 THEN P:='500 W' ELSE IF N_P=56 THEN P:='600 W' ELSE IF N_P=57 THEN P:='700 W' ELSE IF N_P=58 THEN P:='800 W' ELSE IF N_P=59 THEN P:='1000 W'
ELSE
```

```
 IF N_P=60 THEN P:='1500 W';
 POWER_STRING:=P;
END;

{**********}

PROCEDURE CODING_OF_THE_POWER(POWER:BYTE{0 to 60};VAR EM_POWER:STRING);
BEGIN
 EM_POWER:=TRANSFORMATION_N_FINAL_IN_EM_STRING(POWER,1);
END;

{**********}

PROCEDURE DECODING_OF_THE_ANTENNA_GAIN_AND_DIRECTIVITY(EM_ANTENNA:STRING;VAR ANTENNA_GAIN_AND_DIRECTIVITY_STRING:STRING);
VAR N_A,GAIN,DIRECTIVITY:BYTE;
VAR GAIN_STRING,DIRECTIVITY_STRING:STRING;

 {!!!!!!!!!!}
 FUNCTION DISPLAY_GAIN(GAIN:BYTE):STRING;
 BEGIN
  IF GAIN=0 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Vertical whip','Fouet vertical') ELSE
  IF GAIN=1 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Vertical helical','Hélice verticale') ELSE
  IF GAIN=2 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Discone','Discone') ELSE
  IF GAIN=3 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Horizontal Halo','Halo horizontale') ELSE
  IF GAIN=4 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Horizontal Loop','Boucle horizontale') ELSE
  IF GAIN=5 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Magnetic loop','Boucle magnétique') ELSE
  IF GAIN=6 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Dipole','Dipole') ELSE
  IF GAIN=7 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Gain: 4 dBi','Gain: 4 dBi') ELSE
  IF GAIN=8 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Gain: 7 dBi','Gain: 7 dBi') ELSE
  IF GAIN=9 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Gain: 10 dBi','Gain: 10 dBi') ELSE
  IF GAIN=10 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Gain: 13 dBi','Gain: 13 dBi') ELSE
  IF GAIN=11 THEN DISPLAY_GAIN:=AFFICHAGE_TEXTE('Gain: >=16 dBi','Gain: >=16 dBi');
 END;

 {!!!!!!!!!!}
 FUNCTION DISPLAY_DIRECTIVITY(DIRECTIVITY:BYTE):STRING;
 BEGIN
  IF DIRECTIVITY=0 THEN DISPLAY_DIRECTIVITY:='NE' ELSE
  IF DIRECTIVITY=1 THEN DISPLAY_DIRECTIVITY:='E' ELSE
  IF DIRECTIVITY=2 THEN DISPLAY_DIRECTIVITY:='SE' ELSE
  IF DIRECTIVITY=3 THEN DISPLAY_DIRECTIVITY:='S' ELSE
  IF DIRECTIVITY=4 THEN DISPLAY_DIRECTIVITY:='SW' ELSE
  IF DIRECTIVITY=5 THEN DISPLAY_DIRECTIVITY:='W' ELSE
  IF DIRECTIVITY=6 THEN DISPLAY_DIRECTIVITY:='NW' ELSE
  IF DIRECTIVITY=7 THEN DISPLAY_DIRECTIVITY:='N';
 END;

BEGIN
 N_A:=ORDER_CHARACTER_EM_STRING(EM_ANTENNA[1]);{0 to 60}
 IF N_A<=4 THEN {omnidirectional}
 BEGIN
```

```
 GAIN:=N_A;
 ANTENNA_GAIN_AND_DIRECTIVITY_STRING:=DISPLAY_GAIN(GAIN);
 EXTENSION_TO_N_CHARACTERS_WITH_FINAL_SPACES(18,ANTENNA_GAIN_AND_DIRECTIVITY_STRING);
 {WRITELN('Ant ',GAIN,' ',DISPLAY_GAIN(GAIN));}
END ELSE
BEGIN
 N_A:=N_A-5;
 GAIN:=(N_A DIV 8)+5;{5 à 11}
 DIRECTIVITY:=N_A MOD 8;{0 à 7}
 GAIN_STRING:=DISPLAY_GAIN(GAIN);
 EXTENSION_TO_N_CHARACTERS_WITH_FINAL_SPACES(18,GAIN_STRING);
 DIRECTIVITY_STRING:=DISPLAY_DIRECTIVITY(DIRECTIVITY);
 ANTENNA_GAIN_AND_DIRECTIVITY_STRING:=GAIN_STRING+'-> '+DIRECTIVITY_STRING;
END;
END;

{**********}

PROCEDURE CODING_OF_THE_ANTENNA_GAIN_AND_DIRECTIVITY(GAIN,DIRECTIVITY:BYTE;VAR EM_ANTENNA:STRING;VAR SUCCESS:BOOLEAN);
{Antenna gain (0 to 4 omnidirectional/ 5 to 11 directional) and Antenna directivity (0 to 7), in 1 EM character (0 to 60)}
VAR N_A:BYTE;
BEGIN
 IF (GAIN>11) OR (DIRECTIVITY>7) THEN BEGIN SUCCESS:=FALSE; EXIT; END ELSE SUCCESS:=TRUE;

 IF GAIN<=4 THEN N_A:=GAIN {omni directionnal} ELSE
 N_A:=5+(GAIN-5)*8+DIRECTIVITY;{min=5, max=5+6*8+7=60}
 EM_ANTENNA:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_A,1);
END;

{**********}

PROCEDURE DECODING_OF_THE_LATITUDE(EM_LATITUDE:STRING;VAR DEGREES_LAT_GPS,MINUTES_LAT_GPS,THOUSANDTHS_LAT_GPS:WORD;VAR
NORTH_GPS:BOOLEAN;VAR DECIMAL_LATITUDE:DOUBLE;VAR LATITUDE_STRING:STRING);
{Decoding of the latitude Degrees (0 to 89) + Minutes (0 to 59) + Thousandths of minute (0 to 999) +North/South coded into 4 EM characters
LATITUDE_STRING is in the form DD-MM.TTTX with X=N or S, DECIMAL_LATITUDE: decimal degrees i.e. between -89.99999 and 89.99999°}
VAR N_L:DWORD;
VAR COUNTER:BYTE;
VAR A:ARRAY[1..4] OF BYTE;
VAR DEGREES,MINUTES,THOUSANDTHS,NORTH_OR_SOUTH:STRING;
BEGIN
 FOR COUNTER:=1 TO 4 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_LATITUDE[COUNTER]);{0 to 60}
 N_L:=A[1]*POWER(61,3)+A[2]*POWER(61,2)+A[3]*61+A[4];

 {determination of the Latitude}
 IF N_L>=5400000 THEN BEGIN NORTH_GPS:=TRUE; DEC(N_L,5400000); NORTH_OR_SOUTH:='N'; END ELSE BEGIN NORTH_GPS:=FALSE; NORTH_OR_SOUTH:='S'
END;

 DEGREES_LAT_GPS:=N_L DIV 60000; DEC(N_L,DEGREES_LAT_GPS*60000);
 {IF NOMBRE_DE_POSITIONS_BALISE_EM_RECUES MOD 2=1 THEN BEGIN DEC(DEGREES_LAT_GPS,NOMBRE_DE_POSITIONS_BALISE_EM_RECUES);
WRITELN('deg lat ',DEGREES_LAT_GPS); END;//test position GE}
 STR(DEGREES_LAT_GPS:2,DEGREES); IF DEGREES_LAT_GPS<=9 THEN DEGREES[1]:='0';
```

```
 MINUTES_LAT_GPS:=N_L DIV 1000; DEC(N_L,MINUTES_LAT_GPS*1000);
 STR(MINUTES_LAT_GPS:2,MINUTES); IF MINUTES_LAT_GPS<=9 THEN MINUTES[1]:='0';

 THOUSANDTHS_LAT_GPS:=N_L;
 STR(THOUSANDTHS_LAT_GPS:3,THOUSANDTHS);
 IF THOUSANDTHS_LAT_GPS<=9 THEN BEGIN THOUSANDTHS[1]:='0'; THOUSANDTHS[2]:='0'; END ELSE IF THOUSANDTHS_LAT_GPS<=99 THEN
THOUSANDTHS[1]:='0';
 LATITUDE_STRING:=DEGREES+'-'+MINUTES+'.'+THOUSANDTHS+NORTH_OR_SOUTH;

 DECIMAL_LATITUDE:=DEGREES_LAT_GPS+MINUTES_LAT_GPS/60+THOUSANDTHS_LAT_GPS/60000;
 IF NORTH_GPS=FALSE THEN DECIMAL_LATITUDE:=-DECIMAL_LATITUDE;
END;

{**********}

PROCEDURE CODING_OF_THE_LATITUDE(DEGREES_LAT_GPS,MINUTES_LAT_GPS,THOUSANDTHS_LAT_GPS:WORD;NORTH_GPS:BOOLEAN;VAR
EM_LATITUDE:STRING;VAR SUCCESS:BOOLEAN);
{Coding of the latitude Degrees (0 to 89) + Minutes (0 to 59) + Thousandths of minute (0 to 999) +North/South (10 800 000 possibilities) into 4 EM characters (1.38 E7>1.08E7)}
VAR N_L:INT64;
BEGIN
 IF (DEGREES_LAT_GPS>89) OR (MINUTES_LAT_GPS>59) OR (THOUSANDTHS_LAT_GPS>999) THEN BEGIN SUCCESS:=FALSE; EXIT; END ELSE SUCCESS:=TRUE;
 N_L:=DEGREES_LAT_GPS*60000+MINUTES_LAT_GPS*1000+THOUSANDTHS_LAT_GPS;{0 to 5 399 999 Thousandths of minute}
 IF NORTH_GPS=TRUE THEN INC(N_L,5400000);{so the latitude is bewteen 0 to 10799999}
 EM_LATITUDE:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_L,4); //61^4=13845841>10799999
END;

{**********}

PROCEDURE DECODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE(EM_LONGITUDE_ALTITUDE:STRING;ALTITUDE_IN_FEET:BOOLEAN;VAR
DEGREES_LONG_GPS,MINUTES_LONG_GPS,THOUSANDTHS_LONG_GPS:WORD;VAR EAST_GPS:BOOLEAN;VAR DECIMAL_LONGITUDE:DOUBLE;
VAR ALTITUDE_M_GPS:LONGINT;VAR
ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED,ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE:BOOLEAN;VAR
LONGITUDE_STRING,ALTITUDE_STRING:STRING);
{Decoding of the longitude/altitude. LONGITUDE_STRING is in the form DDD-MM.TTTX with X=E or ALTITUDE_STRING is not formatted but must be expressed either in m or in
feet ("ft")
(as for example: "1000 m " if or "3281 ft?", with " " for validated or "?" for not validated,  DECIMAL_LONGITUDE: decimal degrees i.e. between -179.99999 and 179.99999° }
VAR N_L,LONGITUDE:INT64;
VAR ALTITUDE:LONGINT;
VAR COUNTER:BYTE;
VAR A:ARRAY[1..7] OF BYTE;
VAR DEGREES,MINUTES,THOUSANDTHS,EAST_OR_WEST,UNIT_ALTITUDE:STRING;
BEGIN
 FOR COUNTER:=1 TO 7 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_LONGITUDE_ALTITUDE[COUNTER]);{0 to 60}
 N_L:=A[1]*POWER(61,6)+A[2]*POWER(61,5)+A[3]*POWER(61,4)+A[4]*POWER(61,3)+A[5]*POWER(61,2)+A[6]*61+A[7];

 {determination of the Longitude}
 LONGITUDE:=N_L DIV 145496;
 IF LONGITUDE>=10800000 THEN BEGIN EAST_GPS:=TRUE; DEC(LONGITUDE,10800000); EAST_OR_WEST:='E'; END ELSE BEGIN EAST_GPS:=FALSE;
EAST_OR_WEST:='W' END;
```

```
 DEGREES_LONG_GPS:=LONGITUDE DIV 60000; DEC(LONGITUDE,DEGREES_LONG_GPS*60000);
 {IF NOMBRE_DE_POSITIONS_BALISE_EM_RECUES MOD 2=0 THEN BEGIN INC(DEGREES_LONG_GPS,NOMBRE_DE_POSITIONS_BALISE_EM_RECUES);
WRITELN('deg long ',DEGREES_LONG_GPS); END;//test position GE}
 STR(DEGREES_LONG_GPS:3,DEGREES); IF DEGREES_LONG_GPS<=9 THEN BEGIN DEGREES[1]:='0'; DEGREES[2]:='0' END ELSE IF DEGREES_LONG_GPS<=99
THEN DEGREES[1]:='0';

 MINUTES_LONG_GPS:=LONGITUDE DIV 1000; DEC(LONGITUDE,MINUTES_LONG_GPS*1000);
 STR(MINUTES_LONG_GPS:2,MINUTES); IF MINUTES_LONG_GPS<=9 THEN MINUTES[1]:='0';

 THOUSANDTHS_LONG_GPS:=LONGITUDE;
 STR(THOUSANDTHS_LONG_GPS:3,THOUSANDTHS);
 IF THOUSANDTHS_LONG_GPS<=9 THEN BEGIN THOUSANDTHS[1]:='0'; THOUSANDTHS[2]:='0'; END ELSE IF THOUSANDTHS_LONG_GPS<=99 THEN
THOUSANDTHS[1]:='0';

 LONGITUDE_STRING:=DEGREES+'-'+MINUTES+'.'+THOUSANDTHS+EAST_OR_WEST;
 DECIMAL_LONGITUDE:=DEGREES_LONG_GPS+MINUTES_LONG_GPS/60+THOUSANDTHS_LONG_GPS/60000;
 IF EAST_GPS=FALSE THEN DECIMAL_LONGITUDE:=-DECIMAL_LONGITUDE;

 {determination of the Altitude}
 ALTITUDE:=N_L MOD 145496;

 IF ALTITUDE_IN_FEET=TRUE THEN
 BEGIN
  IF LANGUE=1 {English} THEN UNIT_ALTITUDE:='ft' ELSE UNIT_ALTITUDE:='pi';
 END;

 ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE:=TRUE;//initialization;
 IF ALTITUDE<=72747 THEN ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED:=FALSE ELSE BEGIN
ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED:=TRUE; DEC(ALTITUDE,72748) {0 to 72747} END;
 DEC(ALTITUDE,741);
 IF ALTITUDE=72002 THEN
 BEGIN
  ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE:=FALSE;
  ALTITUDE_STRING:='?';
  EXIT;
 END ELSE
 IF ALTITUDE=72001 THEN
 BEGIN
  IF ALTITUDE_IN_FEET=FALSE THEN ALTITUDE_STRING:='>72000 m' ELSE ALTITUDE_STRING:='>236232 '+UNIT_ALTITUDE;
 END ELSE
 IF ALTITUDE=-741 THEN
 BEGIN
  IF ALTITUDE_IN_FEET=FALSE THEN ALTITUDE_STRING:='<-740 m' ELSE ALTITUDE_STRING:='<-2428 '+UNIT_ALTITUDE;
 END ELSE
 BEGIN
  IF ALTITUDE_IN_FEET=FALSE THEN
  BEGIN
   STR(ALTITUDE,ALTITUDE_STRING);
   ALTITUDE_STRING:=ALTITUDE_STRING+' m';
  END ELSE
  BEGIN
```

```
   ALTITUDE:=ROUND(ALTITUDE*3.281);
   STR(ALTITUDE,ALTITUDE_STRING);
   ALTITUDE_STRING:=ALTITUDE_STRING+' '+UNIT_ALTITUDE;
  END;
 END;
 IF ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED=TRUE THEN ALTITUDE_STRING:=ALTITUDE_STRING+' ' ELSE
ALTITUDE_STRING:=ALTITUDE_STRING+'?';
END;

{**********}

PROCEDURE
CODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE(DEGREES_LONG_GPS,MINUTES_LONG_GPS,THOUSANDTHS_LONG_GPS:WORD;EAST_GPS:BOOLEAN;
 ALTITUDE_M_GPS:DOUBLE;ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED,ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE:BOOLEAN;VAR
EM_LONGITUDE_ALTITUDE:STRING;VAR SUCCESS:BOOLEAN);
{Coding of the longitude Degrees (0 to 179) + Minutes (0 to 59) + Thousandths of minute (0 to 999) +East/West (2.16E7 possibilities) + the Altitude into 7 EM characters
(3.1427E12 possibilities)
The altitude (validated or not) is comprised between -740 m and 72000 m.
A bit "V" (ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED) indicates a validated altitude, i.e. a check-sum is present in the $GPGGA frame (because the check-sum is
not compulsory) and this check-sum is validated.
Moreover several individuals values are used, as described below.
The value -741 is to say that the altitude is below -740 m (it's normally impossible to meet such value on Earth which is -378 m)
This wrong altitude can be validated or not (i.e. with bit "V"= o or 1)
The value 72001 is to say that the altitude is above 72000 m (well above the maximum possible for balloons which is 53 km)
This wrong altitude can be validated or not (i.e. with bit "V"= o or 1)
The value 72002 (with bit "V"=0) is to say that the data "altitude" and "number of satellites" are not available, i.e.
 ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE=FALSE if none $GPGGA frame is received or it is not valid (the check-sum being present) during the interval of time
monitored).
The value 72002 (with bit "V"=1) is reserved.
The values 72003 to 72006 (with bit "V"=0 or 1) are reserved.
So the global set of "altitude" values extends from -741 to 72006 i.e. 72748 values
With the "V" bit, it is needed 72748*2=145496 values}
VAR N_L:INT64;
VAR ALTITUDE_M:LONGINT;
BEGIN
 IF (DEGREES_LONG_GPS>179) OR (MINUTES_LONG_GPS>59) OR (THOUSANDTHS_LONG_GPS>999) THEN BEGIN SUCCESS:=FALSE; EXIT; END ELSE
SUCCESS:=TRUE;
 {it is implicitly considered that the altitude could be <740 or >72000}

 {Longitude}
 N_L:=DEGREES_LONG_GPS*60000+MINUTES_LONG_GPS*1000+THOUSANDTHS_LONG_GPS;{0 to 10 799 999 Thousandths of minute}
 IF EAST_GPS=TRUE THEN INC(N_L,10800000);{so the longitude is bewteen 0 to 21599999}

 {Altitude}
 ALTITUDE_M:=ROUND(ALTITUDE_M_GPS);
 IF ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE=FALSE THEN ALTITUDE_M:=72002+741 ELSE
 IF ALTITUDE_M<-740 THEN
 BEGIN
  ALTITUDE_M:=0;
  IF ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED=TRUE THEN INC(ALTITUDE_M,72748);{0 to 145495}
 END ELSE
```

```
 IF ALTITUDE_M>72000 THEN
 BEGIN
  ALTITUDE_M:=72001+741;
  IF ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED=TRUE THEN INC(ALTITUDE_M,72748);{0 to 145495}
 END ELSE
 BEGIN
  ALTITUDE_M:=ALTITUDE_M+741;{0 to 72747}
  IF ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED=TRUE THEN INC(ALTITUDE_M,72748);{0 to 145495}
 END;
 N_L:=N_L*145496+ALTITUDE_M;{0 to 3.142713527E12 <61^7=3.142742836E12-1}
 EM_LONGITUDE_ALTITUDE:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_L,7);
END;

{*********}

PROCEDURE
DECODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES(EM_SPEED_AND_NUM
BER_OF_SATELLITES:STRING;UNIT_OF_SPEED:BYTE;ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED,ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILAB
LE:BOOLEAN;
 VAR SPEED_OVER_GROUND_IN_KNOTS:DOUBLE;VAR Speed_over_ground_Present:BOOLEAN;VAR
NUMBER_OF_QUARTERS_OF_A_CENTURY,NUMBER_OF_VISIBLE_SATELLITES:BYTE;VAR
SPEED_OVER_GROUND_STRING,NUMBER_OF_SATELLITES_STRING:STRING);
{Decoding of the speed over ground (0 to 2971 knots), the number of quarters of century and the number of satellites (3 to 12) coded into 3 EM characters}
{ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED and ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE, are bit managed for the altitude in
DECODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE}
{UNIT_OF_SPEED: 1 if knots, 2 if km/h and 3 if mph}
{SPEED_OVER_GROUND_STRING is not formatted but must be expressed either in knots, km/h or mph (miles per hour), as for example: "1000.0 knots" or "1852.0 km/h" or
"1150.8 mph"
NUMBER_OF_QUARTERS_OF_A_CENTURY is between 0 to 3
NUMBER_OF_SATELLITES_STRING is not formatted but followed by " " for validated or "?" for not validated, as for example: "12?"}
VAR N_G:DWORD;
VAR COUNTER:BYTE;
VAR A:ARRAY[1..3] OF BYTE;
VAR UNIT_SPEED,SPEED_STRING:STRING;
VAR SPEED_OVER_GROUND:DOUBLE;
VAR SPEED:WORD;
BEGIN
 FOR COUNTER:=1 TO 3 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_SPEED_AND_NUMBER_OF_SATELLITES[COUNTER]);{0 to 60}
 N_G:=A[1]*POWER(61,2)+A[2]*61+A[3];

 {determination of the speed over ground}
 SPEED:=N_G DIV 40;//in 1/10th of knots until 299.9 knots and knots from 300 knots
 N_G:=N_G-SPEED*40;
 Speed_over_ground_Present:=TRUE;{initialization}
 IF SPEED=5673 THEN BEGIN Speed_over_ground_Present:=FALSE; SPEED_OVER_GROUND_IN_KNOTS:=0; SPEED_OVER_GROUND_STRING:='?' END ELSE
 IF SPEED=5672 THEN
 BEGIN
  SPEED_OVER_GROUND_IN_KNOTS:=2971;
  IF UNIT_OF_SPEED=1 THEN SPEED_OVER_GROUND_STRING:=AFFICHAGE_TEXTE('>2971 knots','>2971 noeuds') ELSE IF UNIT_OF_SPEED=2 THEN
SPEED_OVER_GROUND_STRING:='>5502 km/h'
  ELSE IF UNIT_OF_SPEED=3 THEN SPEED_OVER_GROUND_STRING:='>3419 mph' ELSE IF UNIT_OF_SPEED=4 THEN SPEED_OVER_GROUND_STRING:='>1528 m/s';
```

```
 END ELSE
 BEGIN
  IF SPEED>=3000 THEN SPEED_OVER_GROUND_IN_KNOTS:=(SPEED-3000)+300 ELSE SPEED_OVER_GROUND_IN_KNOTS:=SPEED/10;
  IF UNIT_OF_SPEED=1 THEN BEGIN SPEED_OVER_GROUND:=SPEED_OVER_GROUND_IN_KNOTS; UNIT_SPEED:=AFFICHAGE_TEXTE('knots','noeuds'); END ELSE
  IF UNIT_OF_SPEED=2 THEN BEGIN SPEED_OVER_GROUND:=SPEED_OVER_GROUND_IN_KNOTS*1.150779; UNIT_SPEED:='mph'; END ELSE
  IF UNIT_OF_SPEED=3 THEN BEGIN SPEED_OVER_GROUND:=SPEED_OVER_GROUND_IN_KNOTS*1.852; UNIT_SPEED:='km/h'; END ELSE
  IF UNIT_OF_SPEED=4 THEN BEGIN SPEED_OVER_GROUND:=SPEED_OVER_GROUND_IN_KNOTS*1.852/3.6; UNIT_SPEED:='m/s'; END;
  IF SPEED>=3000 THEN STR(SPEED_OVER_GROUND:5:0,SPEED_STRING){2971 knots max} ELSE STR(SPEED_OVER_GROUND:5:1,SPEED_STRING);{299.9 knots max}
  SPEED_OVER_GROUND_STRING:=SPEED_STRING+' '+UNIT_SPEED;
  REMOVAL_OF_INITIAL_SPACES(SPEED_OVER_GROUND_STRING);
 END;

 {determination of the number of satellites}
 NUMBER_OF_VISIBLE_SATELLITES:=N_G DIV 4;{0 to 9}
 N_G:=N_G-4*NUMBER_OF_VISIBLE_SATELLITES;
 NUMBER_OF_VISIBLE_SATELLITES:=NUMBER_OF_VISIBLE_SATELLITES+3;{3 to 12}
 IF ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE=FALSE THEN NUMBER_OF_SATELLITES_STRING:='?' ELSE
 BEGIN
  STR(NUMBER_OF_VISIBLE_SATELLITES,NUMBER_OF_SATELLITES_STRING);
  IF ALTITUDE_AND_NUMBER_OF_SATELLITES_VALIDATED=TRUE THEN NUMBER_OF_SATELLITES_STRING:=NUMBER_OF_SATELLITES_STRING+' ' ELSE
NUMBER_OF_SATELLITES_STRING:=NUMBER_OF_SATELLITES_STRING+'?';
 END;

 NUMBER_OF_QUARTERS_OF_A_CENTURY:=N_G;
END;

{*********}

PROCEDURE
CODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES(SPEED_OVER_GROUND_I
N_KNOTS:DOUBLE;Speed_over_ground_Present:BOOLEAN;DATE_STRING:STRING;Date_Present:BOOLEAN;ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE:BO
OLEAN;NUMBER_OF_VISIBLE_SATELLITES:BYTE;
VAR EM_SPEED_AND_NUMBER_OF_SATELLITES:STRING);
{Coding of:
 * the speed over ground (from 0 to 299.9 knots the speed is transformed in 1/10th of knots (so 0 to 2999). From 3000 to 5671 it is in knots, with 3000 for 300 knots. So the
maximum speed is equal to 5671-3000+300=2971 knots (5502 km/h))
 * the number of quarters of a century (0 to 3 so 4 possibilities)for the date cf CODING_OF_THE_TIME_AND_DATE
 * the number of satellites (3 to 12) into 3 EM characters
 The speed over ground in knots (=1,852 km/h=1.150779 mph) is given by a real number. It will transmitted either in the form of tenths of knots up limited to 299.9 knots or in the
form of knots from 300 knots up to 2971 knots
 If no speed over ground available, the speed is equal to 5673
 If the speed is superior to >2371 knots (i.e number=5671), the speed number is equal to 5672
 So there are 5674 values possible (0 to 5673)
 IF Date_Present=FALSE then NUMBER_OF_QUARTERS_OF_A_CENTURY=0 (Note that Date_Present=FALSE will be managed by CODING_OF_THE_TIME_AND_DATE
 The number of satellites: 3 min (absolute minimum) to 12 max (absolute maximum) (10 values)
 If there is no number of satellites  available (i.e. ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE, bit managed  for the altitude in
CODING_OF_THE_LONGITUDE_AND_THE_ALTITUDE), the number of satellites is considered equal to 3 (only for the transmission)
 If there is more than 12 visible satellites, the number of satellites is equal to 13
 The total number of combinations is 5674*4*10=226960 compatible with three EM characters (61^3=226981)}
VAR N_G,N_S:INT64;
VAR SPEED:WORD;//in 1/10th of knots or knots depending of the speed
```

```
VAR NUMBER_OF_QUARTERS_OF_A_CENTURY:BYTE;{0 to 3}
VAR YEAR:DWORD;
VAR COUNTER:BYTE;
BEGIN
 {Speed over ground received converted in 1/10th of knots}
 SPEED:=ROUND(SPEED_OVER_GROUND_IN_KNOTS*10);
 {if the speed is >=300 knots the speed is converted in knots}
 IF SPEED>=3000 THEN SPEED:=ROUND(SPEED_OVER_GROUND_IN_KNOTS-300)+3000;
 IF (Speed_over_ground_Present=FALSE) OR (SPEED<0){in case of...} THEN N_G:=5673 ELSE
 IF SPEED>5671 THEN N_G:=5672 ELSE N_G:=SPEED;{0 to 5671}

 {number of quarters of a century}
 {checking of date + calculation of the representative number}
 IF LENGTH(DATE_STRING)<>6 THEN Date_Present:=FALSE;
 FOR COUNTER:=1 TO 6 DO
 BEGIN
  IF (DATE_STRING[COUNTER]<'0') OR (DATE_STRING[COUNTER]>'9') THEN Date_Present:=FALSE;
 END;
 IF Date_Present=FALSE THEN NUMBER_OF_QUARTERS_OF_A_CENTURY:=0 ELSE
 BEGIN
  YEAR:=(ORD(DATE_STRING[5])-48)*10+(ORD(DATE_STRING[6])-48);//based on 2000
  NUMBER_OF_QUARTERS_OF_A_CENTURY:=YEAR DIV 25;{0 to 3}
  {WRITELN('YEAR ',YEAR,' ',NUMBER_OF_QUARTERS_OF_A_CENTURY);}
 END;

 {Number of satellites (3 to 12 so 0 to 9)}
 IF (ALTITUDE_AND_NUMBER_OF_SATELLITES_AVAILABLE=FALSE) OR (NUMBER_OF_VISIBLE_SATELLITES<3) THEN NUMBER_OF_VISIBLE_SATELLITES:=3
 ELSE IF NUMBER_OF_VISIBLE_SATELLITES>12 THEN NUMBER_OF_VISIBLE_SATELLITES:=12;
 N_S:=NUMBER_OF_VISIBLE_SATELLITES-3;{0 to 9}

 N_G:=N_G*40+N_S*4+NUMBER_OF_QUARTERS_OF_A_CENTURY;{0 to 226979}
 {WRITELN('N_G ',N_G);}
 EM_SPEED_AND_NUMBER_OF_SATELLITES:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_G,3);
END;

{*********}

PROCEDURE DECODING_OF_THE_COURSE_OVER_GROUND(EM_COURSE:STRING;VAR COURSE_OVER_GROUND_IN_DEGREES:DOUBLE;VAR
Course_over_ground_Present:BOOLEAN;VAR COURSE_OVER_GROUND_STRING:STRING);
{Decoding of the course over ground (0 to 3600 1/10th of °) coded into 2 EM characters
SPEED_OVER_GROUND_STRING is not formatted but must be expressed either in °, as for example: "189.8°"}
VAR COUNTER:BYTE;
VAR A:ARRAY[1..2] OF BYTE;
VAR COURSE:WORD;
BEGIN
 FOR COUNTER:=1 TO 2 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_COURSE[COUNTER]);{0 to 60}
 COURSE:=A[1]*61+A[2];

 {determination of the course over ground}
 Course_over_ground_Present:=TRUE;{initialization}
 IF COURSE=3600 THEN BEGIN Course_over_ground_Present:=FALSE; COURSE_OVER_GROUND_IN_DEGREES:=0; COURSE_OVER_GROUND_STRING:='?' END ELSE
```

```
 BEGIN
  COURSE_OVER_GROUND_IN_DEGREES:=COURSE/10;
  STR(COURSE_OVER_GROUND_IN_DEGREES:5:1,COURSE_OVER_GROUND_STRING);
  COURSE_OVER_GROUND_STRING:=COURSE_OVER_GROUND_STRING+' °';
 END;
END;

{*********}

PROCEDURE CODING_OF_THE_COURSE_OVER_GROUND(COURSE_OVER_GROUND_IN_DEGREES:DOUBLE;Course_over_ground_Present:BOOLEAN;VAR
EM_COURSE:STRING);
{Coding of the course over ground (0 to 359.9°, then transformed in 1/10th of °) into 2 EM characters
 The total number of 1/10th of ° is 3600 compatible with thow EM characters (61^2=3721)
 If no course over ground available, the course is equal to 3600}
VAR N_C:INT64;
VAR COURSE:WORD;//in 1/10th of °
BEGIN
 {Course over ground received in 1/10th of ° (0 to 3599)}
 COURSE:=ROUND(COURSE_OVER_GROUND_IN_DEGREES*10);
 IF COURSE=3600 THEN COURSE:=0;

 {If no course over ground available, the course is equal to 3600}
 IF (Course_over_ground_Present=FALSE) OR (COURSE<0)  {in case of...} OR (COURSE>3600) {in case of...} THEN N_C:=3600 ELSE N_C:=COURSE;

 EM_COURSE:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_C,2);
END;

{*********}

PROCEDURE DECODING_OF_THE_TIME_AND_DATE(EM_DATE_TIME:STRING;NUMBER_OF_QUARTERS_OF_A_CENTURY:BYTE;VAR Date_Present:BOOLEAN;VAR
DATE_STRING,TIME_STRING:STRING);
{Decoding of the date and time coded into 5 EM characters
 The GPS UTC time will be displayed in the following form: hh:mm:ss
 NUMBER_OF_QUARTERS_OF_A_CENTURY detemined in
DECODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES
 will permit to determine the exat year (0 to 99)
 The UTC date is displayed in the following form: dd/mm/yy}
VAR COUNTER:BYTE;
VAR A:ARRAY[1..5] OF BYTE;
VAR N_T:INT64;
VAR DATE,TIME:DWORD;
VAR DAY,MONTH,YEAR,HOUR,MINUTE,SECOND:BYTE;
VAR DAY_STRING,MONTH_STRING,YEAR_STRING,HOUR_STRING,MINUTE_STRING,SECOND_STRING:STRING;
BEGIN
 FOR COUNTER:=1 TO 5 DO A[COUNTER]:=ORDER_CHARACTER_EM_STRING(EM_DATE_TIME[COUNTER]);{0 to 60}
 N_T:=A[1]*POWER(61,4)+A[2]*POWER(61,3)+A[3]*POWER(61,2)+A[4]*61+A[5];

 {determination of the date}
 DATE:=N_T DIV 86400;
 Date_Present:=TRUE;{initialization}
 IF DATE=9300 THEN BEGIN Date_Present:=FALSE; DATE_STRING:='?' END ELSE
```

```
 BEGIN
  YEAR:=DATE DIV 372;{0 to 24} DATE:=DATE-YEAR*372;
  YEAR:=YEAR+NUMBER_OF_QUARTERS_OF_A_CENTURY*25;{0 to 24 + 0 to 3 x 25= 0 to 99}
  MONTH:=DATE DIV 31; DAY:=DATE-MONTH*31;
  STR(DAY+1,DAY_STRING); IF DAY+1<=9 THEN DAY_STRING:='0'+DAY_STRING;
  STR(MONTH+1,MONTH_STRING); IF MONTH+1<=9 THEN MONTH_STRING:='0'+MONTH_STRING;
  STR(YEAR,YEAR_STRING);{0 to 99} YEAR_STRING:='20'+YEAR_STRING;
  DATE_STRING:=DAY_STRING+'/'+MONTH_STRING+'/'+YEAR_STRING;
 END;

 {determination of the time}
 TIME:=N_T MOD 86400;
 HOUR:=TIME DIV 3600; TIME:=TIME-HOUR*3600;
 MINUTE:=TIME DIV 60; SECOND:=TIME-MINUTE*60;
 STR(HOUR,HOUR_STRING);  IF HOUR<=9 THEN HOUR_STRING:='0'+HOUR_STRING;
 STR(MINUTE,MINUTE_STRING); IF MINUTE<=9 THEN MINUTE_STRING:='0'+MINUTE_STRING;
 STR(SECOND,SECOND_STRING); IF SECOND<=9 THEN SECOND_STRING:='0'+SECOND_STRING;
 TIME_STRING:=HOUR_STRING+':'+MINUTE_STRING+':'+SECOND_STRING;
END;

{*********}

PROCEDURE CODING_OF_THE_TIME_AND_DATE(DATE_STRING:STRING;TIME_STRING:STRING;Date_Present:BOOLEAN;VAR EM_DATE_TIME:STRING;VAR
SUCCESS:BOOLEAN);
{Coding of the date and time into 5 EM characters. The date is limited to 2048 and based on 2023
 The GPS UTC time (hhmmss) needs: 24*60*60=86400 combinations
 The UTC Date (ddmmyy) needs the year in the present quarter of a century (25 (years)x12 (monthes) x31 (days)= 9300 combinations (year 0 to 24)
 Note that the the number of quarters of a century  is coded in
CODING_OF_THE_SPEED_OVER_GROUND_AND_THE_NUMBER_OF_QUARTERS_OF_A_CENTURY_AND_THE_NUMBER_OF_SATELLITES
 So a year between 0 to 99 is coded with the number of a quarter of a century and the year inside the quarter of a century
 The time is supposed always present, i.e without any time, the frame is supposed invalid and is not taken into account,
 whereas the date is supposed optional.
 So if the date is not present the date number is supposed equal to 9300 so 9301 combinations
 So there is a total of 86400*9301=8.036E8 so it needs 5 EM characters (61^5=8.446E8)}
VAR N_T:INT64;
VAR DAY,MONTH,YEAR,TIME,DATE:DWORD;
VAR COUNTER:BYTE;
BEGIN
 SUCCESS:=FALSE;{initialization}
 {date}
 {checking of date + calculation of the representative number}
 IF LENGTH(DATE_STRING)<>6 THEN Date_Present:=FALSE;
 FOR COUNTER:=1 TO 6 DO
 BEGIN
  IF (DATE_STRING[COUNTER]<'0') OR (DATE_STRING[COUNTER]>'9') THEN BEGIN Date_Present:=FALSE; BREAK; END;
 END;
 DAY:=(ORD(DATE_STRING[1])-48)*10+(ORD(DATE_STRING[2])-48);{1 à 31}
 MONTH:=(ORD(DATE_STRING[3])-48)*10+(ORD(DATE_STRING[4])-48);{1 à 12}
 YEAR:=(ORD(DATE_STRING[5])-48)*10+(ORD(DATE_STRING[6])-48);//based on 2000
 YEAR:=YEAR MOD 25;{0 to 24}
 DATE:=YEAR*372+(MONTH-1)*31+(DAY-1);{min=0, max 9299}
```

```
 {WRITELN('D ',DATE,' ',DAY,' ',MONTH,' ',YEAR);}
 IF DATE>9299 THEN Date_Present:=FALSE;
 IF Date_Present=FALSE THEN DATE:=9300;

 {Time}
 {checking of time + calculation of the representative number}
 IF LENGTH(TIME_STRING)<>6 THEN EXIT;
 FOR COUNTER:=1 TO 6 DO
 BEGIN
  IF (TIME_STRING[COUNTER]<'0') OR (TIME_STRING[COUNTER]>'9') THEN EXIT;
 END;
 TIME:=(ORD(TIME_STRING[1])-48)*36000+(ORD(TIME_STRING[2])-48)*3600+(ORD(TIME_STRING[3])-48)*600+(ORD(TIME_STRING[4])-48)*60+(ORD(TIME_STRING[5])-
48)*10+(ORD(TIME_STRING[6])-48);
 {WRITELN('T ',TIME);}
 IF TIME>86399 THEN EXIT;
 SUCCESS:=TRUE;

 N_T:=DATE*86400+TIME;
 EM_DATE_TIME:=TRANSFORMATION_N_FINAL_IN_EM_STRING(N_T,5);
END;

{**********}

PROCEDURE VERIFICATION_AND_POSSIBLE_TRANSFORMATION_OF_A_CHARACTER_IN_A_FREE_TEXT(CHAR_IN:CHAR;VAR CHAR_OUT:CHAR;VAR
CHAR_EXISTS:BOOLEAN);
{Characters from ASCII 32 to ASCII 95 are allowed , except ^,[ (transformed in "(") and ](transformed in ")"). All letters are in capital}
BEGIN
  {First transformation}
  {a...z in A..Z} {all characters in capitals}
  IF (CHAR_IN>='a') AND (CHAR_IN<='z') THEN CHAR_IN:=CHR(ORD(CHAR_IN)-32) ELSE

  IF (CHAR_IN='Ç') OR (CHAR_IN='ç') THEN CHAR_IN:='C' ELSE

  IF (CHAR_IN='ü') OR (CHAR_IN='û') OR (CHAR_IN='ù') OR
     (CHAR_IN='Ü') OR (CHAR_IN='ú')  THEN CHAR_IN:='U' ELSE

  IF (CHAR_IN='é') OR (CHAR_IN='ê') OR
     (CHAR_IN='æ') OR (CHAR_IN='Æ') OR (CHAR_IN='ë') OR
     (CHAR_IN='è') OR (CHAR_IN='É')  THEN CHAR_IN:='E' ELSE

  IF (CHAR_IN='â') OR (CHAR_IN='ä') OR (CHAR_IN='à') OR
     (CHAR_IN='å') OR (CHAR_IN='Ä') OR (CHAR_IN='Å')
     OR (CHAR_IN='á')  OR (CHAR_IN='ª') THEN CHAR_IN:='A' ELSE

  IF (CHAR_IN='ï') OR (CHAR_IN='î') OR (CHAR_IN='ì') OR
     (CHAR_IN='í') THEN CHAR_IN:='I' ELSE

  IF (CHAR_IN='ô') OR (CHAR_IN='ö') OR (CHAR_IN='ò') OR
    (CHAR_IN='Ö') OR (CHAR_IN='ó') OR (CHAR_IN='º')  THEN CHAR_IN:='O' ELSE

  IF (CHAR_IN='ÿ') THEN CHAR_IN:='Y' ELSE
```

```
 IF (CHAR_IN='ø') OR (CHAR_IN='Ø') THEN CHAR_IN:='0' ELSE

 IF (CHAR_IN='×') THEN CHAR_IN:='X' ELSE
 IF (CHAR_IN='ƒ') THEN CHAR_IN:='F' ELSE
 IF (CHAR_IN='¿') THEN CHAR_IN:='?' ELSE
 IF (CHAR_IN='©') OR (CHAR_IN='¢') THEN CHAR_IN:='C' ELSE
 IF (CHAR_IN='Ð') THEN CHAR_IN:='D' ELSE

 IF (CHAR_IN='ñ') OR (CHAR_IN='Ñ') THEN CHAR_IN:='N' ELSE

 IF CHAR_IN='[' THEN CHAR_IN:='(' ELSE
 IF CHAR_IN=']' THEN CHAR_IN:=')';

 {characters accepted: from 32 to 95 except "^", "[" and "]"}
 IF ((CHAR_IN>=' ') AND (CHAR_IN<='Z')) OR (CHAR_IN='_') OR (CHAR_IN='\') THEN
 BEGIN
  CHAR_OUT:=CHAR_IN;
  CHAR_EXISTS:=TRUE;
 END ELSE
 BEGIN
  CHAR_OUT:=CHR(0);
  CHAR_EXISTS:=FALSE;
 END;
END;

{*********}

PROCEDURE CONTROL_OF_A_FREE_TEXT(FREE_TEXT:STRING;VAR EM_FREE_TEXT:STRING);
{Control a free text. The 6 bits characters used are part of the ASCII characters (used for occidental languages), from CHR(32) to CHR(95), except
the characters "^", "[", "]". The coded characters are in EM_FREE_TEXT}
VAR COUNTER:BYTE;
VAR CHAR_IN,CHAR_OUT:CHAR;
VAR CHAR_EXISTS:BOOLEAN;
BEGIN
 EM_FREE_TEXT:='';
 {verification of the characters}
 FOR COUNTER:=1 TO LENGTH(FREE_TEXT) DO
 BEGIN
  CHAR_IN:=FREE_TEXT[COUNTER];
  VERIFICATION_AND_POSSIBLE_TRANSFORMATION_OF_A_CHARACTER_IN_A_FREE_TEXT(CHAR_IN,CHAR_OUT,CHAR_EXISTS);
  IF CHAR_EXISTS=TRUE THEN EM_FREE_TEXT:=EM_FREE_TEXT+CHAR_OUT;
 END;
END;

{**********}

FUNCTION CHECKSUM_EM_FRAME(FRAME:STRING):CHAR;
{It is calculated the checksum on the frame on one EM character, thanks to a XOR bewteen all characters of the frame}
VAR COUNTER,CODE:BYTE;
VAR EM_CHAR:CHAR;
```

```
begin
 CODE:=0;{initialization}
 FOR COUNTER:=1 TO LENGTH(FRAME) DO CODE:=CODE XOR ORD(FRAME[COUNTER]);{0 to 127}
 CODE:=CODE MOD 61;{0 to 60}
 EM_CHAR:=CHR(32+CODE);{32 min to 32+60=92}
 IF EM_CHAR='[' THEN EM_CHAR:='_';
 CHECKSUM_EM_FRAME:=EM_CHAR;
end;
```