EXAMPLE

Example calling packmsg(msg,dat) using your message (i.e. msg = “TF2MSN KA1OVM FN42 “ (padded with spaces to 22 characters)

It is not a ‘CQ ‘ message, so i= position of 1st space i.e. 7

ia = 7 and c1 = ‘TF2MSN ’ (again padded automatically to 12 bytes/chars)

i = position of next space i.e. 14

ib = 14 and c2 = ‘KA1OVM ‘ (again padded to 12 bytes/chars)

i = position of next space i.e. 10

ic = 19 and c3 is set = ‘FN42 ‘ but since c is only 4 chars c = ‘FN42’

Getpfx is called with ‘TF2MSN ’, returning k1 = 0

“since callsign contains no slash, it pass though and return 0”

Packcall is called with ‘TF2MSN ’, returning nc1 = 208499153 and text1 = false

“text=false. since 3rd char of callsign is a digit, tmp=callsign. Tmp is a valid callsign, so ncall is computed as:

ncall = nchar(‘T’) or 29

ncall = 36 \* 29 + nchar(‘F’) or 1044 + 15 or 1059

ncall = 10 \* 1059 + nchar(‘2’) or 10590 + 2 or 10592

ncall = 27 \* 10592 + nchar(‘M’) or 285984 + 22 or 286006

ncall = 27 \* 286006 + nchar(‘S’) or 7722162 + 28 or 7722190

ncall = 27 \* 7722190 + nchar(‘N’) or 208499130 + 23 or 208499153

ncall = 208499153 decimal or 1100011011010111000111010001 binary and text = false

Getpfx is called with ‘KA1OVM ‘, returning k2 = 0

“since callsign contains no slash, it pass though and return 0”

Packcall is called with ‘KA1OVM ‘, returning nc2 = 143723938 and text2 = false

“text=false. since 3rd char of callsign is a digit, tmp=callsign. Tmp is a valid callsign, so ncall is computed as:

ncall = nchar(‘K’) or 20

ncall = 36 \* 20 + nchar(‘A’) or 720 + 10 or 730

ncall = 10 \* 730 + nchar(‘1’) or 7300 + 1 or 7301

ncall = 27 \* 7301 + nchar(‘O’) or 197127 + 24 or 197151

ncall = 27 \* 197151 + nchar(‘V’) or 5323077 + 31 or 5323108

ncall = 27 \* 5323108 + nchar(‘M’) or 143723916 + 22 or 143723938

ncall = 143723938 decimal or 1000100100010000110110100010 binary and text = false

packgrid is called with c3=’FN42’ returning ng=22632 and text3=false

skips special cases of – or R-

verifies 4 character sequence is <A-Z><A-Z><0-9><0-9>

calls grid2deg(‘FN42mm’,dlong,dlat) returns dlong=71 and dlat=42

ng = (71+180)/2 \* 180 + (42+90) or 22500 + 132 (integers are truncated) or 22632

Dat[1] = AND(RSHIFT(208499153,22),63) or 49 or 110001

Dat[2] = AND(RSHIFT(208499153,16),63) or 45 or 101101

Dat[3] = AND(RSHIFT(208499153,10),63) or 28 or 011100

Dat[4] = AND(RSHIFT(208499153, 4),63) or 29 or 011101

Dat[5] = (4\*AND(208499153,15))+AND(RSHIFT(143723938,26),3) or (4\*1)+2 or 6 or 000110

Dat[6] = AND(RSHIFT(143723938,20),63) or 9 or 001001

Dat[7] = AND(RSHIFT(143723938,14),63) or 4 or 000100

Dat[8] = AND(RSHIFT(143723938, 8),63) or 13 or 001101

Dat[9] = AND(RSHIFT(143723938, 2),63) or 40 or 101000

Dat[10] = (16\*AND(143723938,3))+AND(RSHIFT(22632,12),15) or (16\*2)+5 or 37 or 100101

Dat[11] = AND(RSHIFT(22632,6),63) or 33 or 100001

Dat[12] = AND(22632,63) or 40 or 101000

So a msg = “TF2MSN KA1OVM FN42 “ becomes 49,45,28,29,6,9,4,13,40,37,33,40

General Notes:

* The default variable type in Fortran, unless declared otherwise, is an integer, which is a 4 byte integer –From -2147483648 to +2147483647 or 1 sign bit and 31 number bits.
* All function or subroutine parameters are passed by reference and so any changes to the variable in the function or subroutine will change the content of the variable in the calling routine.

Packmsg (msg “Input”, dat “output”) where msg = 22 byte/character string. NOTE: character variables are padded automatically with spaces to their length if needed.

dat = array of 12 32-bit integers

c1,c2 are 12 characters (bytes) long

c3 is 4 characters long

grid6 is 6 characters long

text1,text2, text3 are logical (true/false)

Step 1) msg = uppercase(msg)

Step 2) IF msg[1:3] = ‘CQ ‘ THEN

i=3

“if msg is ‘CQ ‘ check for reply frequency”

IF msg[4:6] is an integer “all 3 digits/chars between 0-9)” THEN

i=7

ENDIF

Jump to Step 3

ENDIF

i = position of the 1st space character

OR (i=22 “if no spaces in msg” and Jump to Step 4)

Step 3) ia = i

c1 = msg[1:ia-1] “c1 gets the text before the space”

i = position of next space after position ia

OR (i=22 and Jump to Step 4)

ib = i

c2 = msg[ia+1:ib-1] “c2 gets the next text after position ib”

i = position of next space after position ib

OR (i=22 and Jump to Step 4)

ic = i

c3 = ‘ ‘ “c3 set to 4 spaces – technically not needed since padded automatically”

IF (ic >= ib+1) THEN c3 = msg[ib+1:ic]

IF (c3 = ‘000 ‘) THEN c3 = ‘ ‘ “c3 is set to 4 spaces”

Call getpfx1(c1 “input”,k1 “output”)

Call packcall(c1 “input”,nc1,text1 “outputs”)

Call getpfx1(c2 “input”,k2 “outputs”)

Call packcall(c2 “input”,nc2,text2 “outputs”)

“if either k1 or k2 is less than 0 or either one equals 0 then jump to step 4”

IF (k1 < 0) OR (k2 < 0) OR (k1\*k2 <> 0) THEN Jump to Step 4

IF (k2 > 0) THEN k2=k2+450

k = MAX(k1,k2)

IF (k > 0) THEN

Call k2grid(k “input”,grid6 “output”)

c3 = grid6

ENDIF

Call packgrid(c3 “input”,ng,text3 “outputs”)

IF ((NOT text1) AND (NOT text2) AND (NOT text3)) THEN Jump to Step 5

Step 4 “Step 4 is for handling messages ‘to treated as plain text’”

Call packText(msg “input”,nc1,nc2,ng “outputs”)

ng = ng + 32768 “1000000000000000 binary – 16th bit set to 1”

Step 5 “Encode the data into 6-bit words”

“RSHIFT(a1,a2) shifts a1 arithmetically right by a2 bits. Equivalent to the C >> operator.”

“63 is 111111 binary, 3 is 11 binary, and 15 is 1111 binary”

Dat[1] = AND(RSHIFT(nc1,22),63) “6 bits”

Dat[2] = AND(RSHIFT(nc1,16),63) “6 bits”

Dat[3] = AND(RSHIFT(nc1,10),63) “6 bits”

Dat[4] = AND(RSHIFT(nc1, 4),63) “6 bits”

Dat[5] = (4\*AND(nc1,15))+AND(RSHIFT(nc2,26),3) “4+2 bits”

Dat[6] = AND(RSHIFT(nc2,20),63) “6 bits”

Dat[7] = AND(RSHIFT(nc2,14),63) “6 bits”

Dat[8] = AND(RSHIFT(nc2, 8),63) “6 bits”

Dat[9] = AND(RSHIFT(nc2, 2),63) “6 bits”

Dat[10] = (16\*AND(nc2,3))+AND(RSHIFT(ng,12),15) “2+4 bits”

Dat[11] = AND(RSHIFT(ng,6),63)

Dat[12] = AND(ng,63)

Return array Dat

packText(msg “input”, nc1,nc2,nc3 “outputs”)

msg is 13 characters/bytes – passing a longer string will truncate

c is a constant containing ‘0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ +-./?’ (excluding the single quotes)

nc1,nc2,nc3 initialized to 0

Step 1) LOOP each of the 5 characters of msg i.e. msg[1:5]

j = position of the character in the constant c or c = 37 (space)

j = j-1

nc1 = 42 \* nc1 + j “base 42 math (the position of each characters is shifted by 42”

ENDLOOP

Step 2) LOOP each of the next 5 characters of msg i.e. msg[6:10]

j = position of the character in the constant c or c = 37 (space)

j = j-1

nc2 = 42 \* nc2 + j

ENDLOOP

Step 3) LOOP each of the last 3 characters of msg i.e. msg[11:13]

j = position of the character in the constant c or c = 37 (space)

j = j-1

nc3 = 42 \* nc3 + j

ENDLOOP

Step 5) “Since the last chunk is only 3 characters, only 17 bits have been used, so 1 bit each is moved to nc1 and nc2”

nc1 = nc1 + nc1

IF (AND(nc3,32768) <> 0) THEN nc1=nc1+1 “32768 = 1000000000000000 16-bits”

nc2 = nc2 + nc2

IF (AND(nc3,65536) <> 0) THEN nc2=nc2+1 “65536 = 10000000000000000 17-bits”

nc3 = AND(nc3,32767) “32767 = 0111111111111111 16 bits”

return nc1,nc2,nc3

This routine is encoding a number, k, that represents the found positions of a portion of the input string in a prefix array and a portion in a suffix array.

getpfx1(callsign “input”, k “output”) where callsign is 12 characters, k is an integer. callsign is a string of non-space characters followed by all spaces (due to padding)

c is defined as 4 characters long

“includes the file ‘pfx.f’ which defined 4 ‘constants’ NZ=338, NZ2=11, sfx an 11 element array of single characters = ‘P0123456789’ and pfx and array of 338 5-character elements which appear to be call sign prefixes such as:

‘1A ‘, ‘3D2 ‘, etc.”

iz = position of last non-space character in callsign

islash = position of a ‘/’ in callsign OR 0 if no ‘/’

k = 0

c = ‘ ‘ “4 spaces”

Step 1) IF (a slash is found i.e. non-zero AND (islash <= 4 OR (islash = 5 AND iz >= 8 “i.e. the input is 8 chars or more”)) THEN

“X/\*\*\*\*\*\*\*\*\*\*” X=nonspace character \*=space or nonspace character

“XX/\*\*\*\*\*\*\*\*\*” “these are callsign patterns matched”

“XXX/\*\*\*\*\*\*\*\*”

“XXXX/XXX\*\*\*\*“

c = callsign[1:islash-1] “get the characters before the ‘/’”

callsign = callsign[islash+1:iz] “set callsign = text after ‘/’”

IF (1st 4 chars of c found in array pfx) THEN

i = found element number “i.e. index of the array”

k = i “i is a number from 1 to 338”

Jump to Step 2

ENDIF

ELSE IF (islash > 5 OR (islash = 5 AND iz = 6) THEN

“XXXX/X “ X=nonspace character \*=space or nonspace character

“XXXXX/\*\*\*\*\*\*” “these are callsign patterns matched”

c = callsign[islash+1:iz] “get the chars after the /”

callsign = callsign[1:islash-1] “set callsign to text before /”

IF (1st char of c found in array sfx) THEN

i = found element number “i.e. index of the array”

k = 400 + I “i is a number from 1 to 11”

Jump to Step 2

ENDIF

ENDIF

“If no slash is present in callsign, the program goes to step 2”

Step 2) IF (islash <> 0 AND k = 0) THEN k = -1 “set k to -1 if callsign contained a slash but the components were not found in the arrays”

Return “if no / in callsign, k = 0 and callsign is unchanged”

Packcall(callsign “input”,ncall,text “outputs”) where callsign is 6 characters, ncall is an integer and text is a logical (true/false)

Constant NBASE = 37\*36\*10\*27\*27\*27 (or 262177560 or 1111101000001000001100011000 binary (28 bits)

Callsign is 6 characters

C is 1 char

Tmp is 6 characters

Text is a logical (true/false)

Text = false

Step 1) IF (callsign[1:3] = ‘CQ ‘) THEN

ncall = NBASE + 1 “so 262177561 or 1111101000001000001100011001”

IF (callsign[4:6] is a number) THEN

nfreq = TextToInteger(callsign[4:6]) “covert from 3 chars to #”

ncall = NBASE + 3 + nfreq

ENDIF

return “exit function with text = false and ncall set”

ELSEIF (callsign[1:4] = ‘QRZ ‘) THEN “input starts with ‘QRZ ‘”

ncall = NBASE + 2 “so 262177562 or 1111101000001000001100011010”

return “exit function with text = false and ncall set”

ENDIF

Step 2) tmp = ‘ ‘ “6 spaces”

IF (3rd character of callsign is a digit (0-9)) THEN

tmp = callsign

ELSEIF (2nd character of callsign is a digit (0-9)) THEN

IF (6th character of callsign <> space) THEN

text = true

return “exit function with text = true and ncall zero”

ENDIF

tmp = ‘ ‘ concatenated with callsign “shifts text of callsign to right by 1 space, truncates right most character”

ELSE

text = true

return “exit function with text = true and ncall zero”

ENDIF

Step 3) “get here when callsign’s 3rd char is digit OR (2nd char is digit and 6th char = space)”

tmp = uppercase(tmp)

“code below seems to validate that callsign is a valid call sign”

IF tmp[1] is 0-9,A-Z or space THEN n1=1 ELSE n1=0

IF tmp[2] is 0-9,A-Z THEN n2=1 ELSE n2=0

IF tmp[3] is 0-9 THEN n3=1 ELSE n3=0

IF tmp[4] is A-Z or space THEN n4=1 ELSE n4=0

IF tmp[5] is A-Z or space THEN n5=1 ELSE n5=0

IF tmp[6] is A-Z or space THEN n6=1 ELSE n6=0

IF (n1+n2+n3+n4+n5+n6 <> 6) THEN “if callsign does not match the pattern above then exit with text = true”

text = true

return

ENDIF

Step 4) “nchar is a custom function that encodes the character as a number between 0 and 36 ordered as 0-9A-Z<space>, so the character ‘0’ is zero and ‘9’ is nine and ‘A’ is 10 and so on with space = 36”

ncall = nchar(tmp[1]) “1st char is 36 possible values”

ncall = 36\*ncall + nchar(tmp[2]) “x36 since 2nd char is 35 possible”

“NOTE: properly the above should be x35, not 36 since space is not valid for char 2 which means a 35 character set not 36”

ncall = 10\*ncall + nchar(tmp[3]) “x10 since 3rd char is 10 possible”

ncall = 27\*ncall + nchar(tmp[4])-10 “x27 since 4th char is 27 values”

ncall = 27\*ncall + nchar(tmp[5])–10 “x27 since 5th char is 27 values”

ncall = 27\*ncall + nchar(tmp[6])–10 “x27 since 6th char is 27 values”

return ncall, text

k2grid(k “input”,grid6 “output”)

grid6 – 6 character string

nlong = 2\*mod((k-1)/5,90)-179 “modulo 90 to get longitude”

if (k > 450) nlong=nlong+180

nlat = mod(k-1,5) + 85 “modula 5 to get latitude”

dlat = nlat

dlong = nlong

call deg2grid(dlong,dlat,grid)

return

deg2grid(dlong0,dlat,grid)

real dlong !West longitude (deg)

real dlat !Latitude (deg)

character grid\*6

dlong = dlong0

IF (dlong < -180.0) THEN dlong=dlong+360.0

IF (dlong > 180.0) THEN dlong=dlong-360.0

Step 1) “Convert to units of 5 min of longitude, working east from 180 deg.”

nlong=60.0\*(180.0-dlong)/5.0

n1=nlong/240 “20-degree field”

n2=(nlong-240\*n1)/24 “2 degree square”

n3=nlong-240\*n1-24\*n2 “5 minute subsquare”

grid[1] = NumToChar(CharToNum('A')+n1)

grid[3] = NumToChar(CharToNum('0')+n2)

grid[5] = NumToChar(CharToNum('a')+n3)

Step 2) “Convert to units of 2.5 min of latitude, working north from -90 deg.”

nlat=60.0\*(dlat+90)/2.5

n1=nlat/240 !10-degree field

n2=(nlat-240\*n1)/24 !1 degree square

n3=nlat-240\*n1-24\*n2 !2.5 minute subsquare

grid[2] = NumToChar(CharToNum('A')+n1)

grid[4] = NumToChar(CharToNum('0')+n2)

grid[6] = NumToChar(CharToNum('a')+n3)

return grid

packgrid(grid “input”, ng,text “output”)

CONSTANT NGBASE = 180\*180 “32400 or 111111010010000”

grid – 4 characters

text – logical (true/false)

text = false

IF (grid = ‘ ‘ “4 spaces”) THEN Jump to Step 3

“Test for VK/ZL signal report”

Step 1) IF (grid[1] = ‘-‘) THEN

n = StringToNum(grid[2:3]) “n=0 if char 2-3 of grid is not a number”

ng = NGBASE + 1 + n

return ng, text

ELSEIF (grid[1:2] = “R-“) THEN

n = StringToNum(grid[3:4])

IF (n=0) THEN Jump to Step 3

ng = NGBASE + 31 + n

return ng, text

ENDIF

Step 2) IF (grid[1] <> A-Z) THEN text=true

IF (grid[2] <> A-Z) THEN text=true

IF (grid[3] <> 0-9) THEN text=true

IF (grid[4] <> 0-9) THEN text=true

IF (text) THEN

return ng,text

ENDIF

call grid2deg(CONCATINATE(grid,’mm),dlong,dlat)

long = dlong

lat = dlat + 90

ng = ((long+180)/2)\*180 + lat

Step 3) ng = NGBASE + 1

return ng, text

“grid location as long,lat is being packed as base 180 math? Seems like all encryption in this entire program is based upon encrypting letters by position and numbers by the base of the range of the number.”