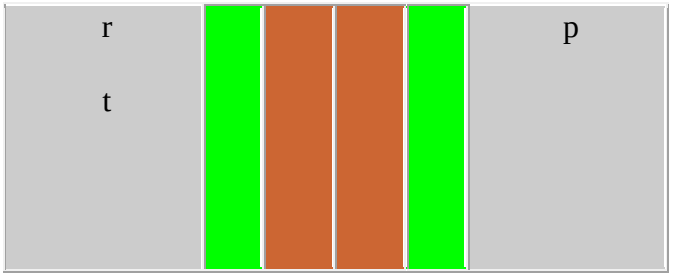
The PDF417 code

This code is part of 2 dimensional code family, it's in fact a code with several rows which can encode up to 2700 bytes what explain its name of "Portable Document File". The encoding is done in two stages: first the data are converted to "codeword" (High level encoding) then those are converted to bars and spaces patterns. (Low level encoding) Moreover an error correction system with several levels is included, it allows reconstituting badly printed, erased, fuzzy or torn off data. In the continuation of this talk, the word "codeword" will be shortened into CW.

The general structure.

- The width of the finest bar is called the module.
- Thereafter a bar module is symbolized by "1" and a space module by "0".
- The code is composed of 3 to 90 rows.
- A row is made of 1 to 30 data columns and its width goes from 90 to 583 modules with the margins.
- Maximum number of CW in bar codes: 928 including 925 for the data. (1 for the length descriptor and 2 at least for the error correction.)
- If necessary a mechanism named "Macro PDF417" allows distributing more data on several bar codes.
- There are 929 CW including 900 for the data, they are numbered from 0 to 928.
- The errors correction levels goes from 0 to 8. The correction comprises 2 (on level 0) to 512 (on level 8) CW.
- The row consists of a start character, a left side CW, 1 to 30 data CW, a right side CW and a stop character. There must be a white margin of at least 2 modules on each side.
- CW of padding (We'll use by example "900") can be intercalated between data and correction CW, those must be located at the end.
- First CW indicates CW total number of the code including: data, CW of stuffing and itself but excluding CW correction.
- Sample of code with 14 data CW, a 15th CW indicate CW number, one padding CW and 4 correction CW. (Level 1)

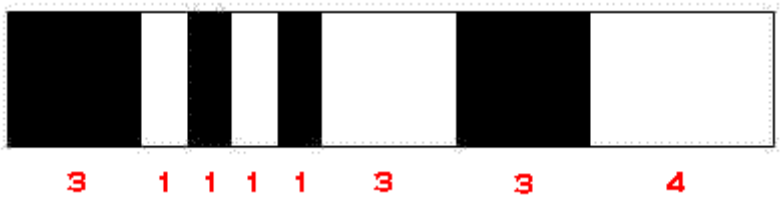
S	L1	D15	D14	R1	S
	L2	D13	D12	R2	
	L3	D11	D10	R3	
	L4	D9	D8	R4	
	L5	D7	D6	R5	
	L6	D5	D4	R6	
t	L7	D3	D2	R7	t
	L8	D1	D0	R8	
a	L9	C3	C2	R9	o
	L10	C1	C0	R10	



D15 = length descriptor (16 in this sample)
D0 = padding
D1 a D14 = data
L1 a L10 = left side CW
R1 a R10 = right side CW
C0 a C3 = error correction, level 1

Low level encoding.

- Each CW is made of 17 modules, containing 4 bars and 4 spaces (Name code comes from there !) and it start by a bar. Bars and spaces width is 1 to 6 modules. (Except for start and stop characters) Sample :



that is to say the following formulation : 111 0 1 0 1 000 111 0000

- Start character is : 11111111 0 1 0 1 0 1 000
- Stop character is : 1111111 0 1 000 1 0 1 00 1 (Here there is a 5th bar, thus 18 modules.)
- There are 3 distinct tables for encoding the 929 CW.
- The 3 tables giving the patterns for the 929 CW start like this :

First table	*Second table*	*Third table*
111 0 1 0 1 0 111 000000	11111 0 1 0 1 0 11 00000	11 0 1 0 1 0 11111 00000
1111 0 1 0 1 0 1111 0000	111111 0 1 0 1 0 111 000	111 0 1 0 1 0 111111 000
11111 0 1 0 1 0 11111 00	1111 0 1 0 1 00 1 000000	1 0 1 0 1 00 1111 000000
111 0 1 0 1 00 111 00000	11111 0 1 0 1 00 11 0000	11 0 1 0 1 00 11111 0000
.....	.....	.....

- Each row use only one encoding table, this table will be used again 3 rows further. Sample : Row 1 --> table 1, row 2 --> table 2, row 3 --> table 3, row 4 --> table 1, ...etc. We have the formula : table number = ((row number MOD 3) * 3)

High level encoding.

- Thereafter we'll use operators: + --> addition, x --> multiplication, \ --> integer division, MOD --> remainder of the integer division.
- Data are compacted into CW according to 3 modes which are conceived according to their efficiency compared to the data type. Default mode is "Text" mode.

Compaction mode	*Data to encode*	*Rate compaction*
"Byte"	ASCII 0 to 255	1.2 byte per CW
"Text"	ASCII 9, 10, 13 & 32 a 127	2 characters per CW
"Numeric"	Only digits 0 to 9	2.9 digits per CW

- The CW number 900 to 928 have special meaning, some enable to switch between modes in order to optimise the code.

CW number :	*Function*
900	Switch to "Text" mode
901	Switch to "Byte" mode
902	Switch to "Numeric" mode
903 a 912	Reserved
913	Switch to "Octet" only for the next CW
914 a 920	Reserved
921	Initialization
922	Terminator codeword for Macro PDF control block
923	Sequence tag to identify the beginning of optional fields in the Macro PDF control block
924	Switch to "Byte" mode (If the total number of byte is multiple of 6)
925	Identifier for a user defined Extended Channel Interpretation (ECI)
926	Identifier for a general purpose ECI format
927	Identifier for an ECI of a character set or code page
928	Macro marker CW to indicate the beginning of a Macro PDF Control Block

- The "Text" mode have 4 sub-modes :
  - Uppercase
  - Lowercase
  - Mixed : Numeric and punctuation
  - Punctuation

  The default sub-mode is "Uppercase", in this sub-mode 2 characters are encoded in each CW, here is characters tables :

Value	*Uppercase*	*Lowercase*	*Mixed*	*Punctuation*
0	A	a	0	;
1	B	b	1	<
2	C	c	2	>
3	D	d	3	@
4	E	e	4	[
5	F	f	5	\
6	G	g	6	]
7	H	h	7	_
8	I	I	8	` (Quote)
9	J	j	9	~

10	K	k	&	!
11	L	l	CR	CR
12	M	m	HT	HT
13	N	n	,	,
14	O	o	:	:
15	P	p	#	LF
16	Q	q	-	-
17	R	r	.	.
18	S	s	$	$
19	T	t	/	/
20	U	u	+	□g
21	V	v	%	|
22	W	w	*	*
23	X	x	=	(
24	Y	y	^	)
25	Z	z	PUN	?
26	SP	SP	SP	{
27	LOW	T_UPP	LOW	}
28	MIX	MIX	UPP	' (Apostrophe)
29	T_PUN	T_PUN	T_PUN	UPP

6 switches are included in these tables, they allow to change the sub-mode :
UPP : switch to "Uppercase"
LOW : switch to "Lowercase"
MIX : switch to "Mixed"
PUN : switch to "Punctuation"
T_UPP : switch to "Uppercase" only for next character
T_PUN : switch to "Punctuation" only for next character
Each CW encode 2 characters ; if C_1 and C_2 are the values of the two characters, CW value is : $C_1 \times 30 + C_2$
If it remains an alone character, we add to it a padding switch, for instance T_PUN.

Sample, sequence to encode : Super !

S : 18, LOW : 27, u : 20, p : 15, e : 4, r : 17, SPACE : 26, T_PUN : 29, ! : 10
that is 9 characters, we'll add a T_PUN for the padding.
$CW_1 = 18 \times 30 + 27 = 567$
$CW_2 = 20 \times 30 + 15 = 615$
$CW_3 = 4 \times 30 + 17 = 137$
$CW_4 = 26 * 30 + 29 = 809$
$CW_5 = 10 \times 30 + 29 = 329$
The sequence is consequently : 567, 615, 137, 809, 329

- The "Byte" mode allows to encode 256 different bytes, that is the entire extended ASCII table.
  If the byte number is a multiple of 6, we use the 924 CW to switch to "Byte" mode; if the byte number is not a multiple of 9 we use 901 CW for switching.
  Coding consists has to transform 6 bytes in base 256 to 5 CW in base 900. To carry out the conversion :
  - Take the bytes by group of 6; let X_5 to X_0 their decimal values. (X_0 is the less significant character)
  - Compute the sum $S = X_5 \times 256^5 + X_4 \times 256^4 + X_3 \times 256^3 + X_2 \times 256^2 + X_1 \times 256 + X_0$

- Let's compute the CWs : CW_0 = S MOD 900, new value of S : S = S \ 900, CW_1 = S MOD 900 ... and so forth to CW4 (CW_0 is the less significant CW)
- Bytes which remains after the conversion of the groups of 6 are taken just as they are : 1 byte = 1 CW of same value.

Sample 1 : word to encode : alcool

The sequence of bytes (in ASCII) is : 97, 108, 99, 111, 111, 108
S = 97 x 256^5 + 108 x 256^4 + 99 x 256^3 + 111 x 256^2 + 111 x 256 + 108 = 107 118 152 609 644
CW_0 = 107 118 152 609 644 MOD 900 = 244
S = 107 118 152 609 644 \ 900 = 119 020 169 566
CW_1 = 119 020 169 566 MOD 900 = 766
S = 119 020 169 566 \ 900 = 132 244 632
CW_2 = 132 244 632 MOD 900 = 432
S = 132 244 632 \ 900 = 146 938
CW3 = 146 938 MOD 900 = 238
S = 146 938 \ 900 = 163
CW4 = 163 MOD 900 = 163
The sequence including the switch is consequently : 924, 163, 238, 432, 766, 244

Sample 2 : word to encode : alcoolique

The sequence of bytes (in ASCII) is : 97, 108, 99, 111, 111, 108, 105, 113, 117, 101
The first 6 bytes are coded like above and we add 105, 113, 117 and 101
The sequence including the switch is consequently : 901, 163, 238, 432, 766, 244, 105, 113, 117, 101

- The "Numeric" mode is a conversion from base 10 to base 900
  To carry out this conversion :
  - Take the digits by group of 44 (or less for the rest)
  - Add the digit 1 ahead, it will be removed by the decoding procedure
  - Make the change of base like for the "Byte" mode
  - Each group of 44 digits give 15 CWs
  - The smallest groups give a number of CW witch is : Number of digits \ 3 + 1
    Sample : 10 digits give 10 \ 3 + 1 = 3 + 1 = 4 CWs

Sample, sequence to encode : 01234

There will be thus 5 \ 3 + 1 = 2 CW
Add a "1" ahead : 101234
CW_0 = 101 234 MOD 900 = 434
S = 101 234 \ 900 = 112
CW_1 = 112 MOD 900 = 112
The sequence is consequently : 112, 434

- Left and right side CWs are computed according to the table used for the actual row.
  To obtain the CW value, make the following calculation: (Row Number \ 3) x 30 + X with X taken in the following table. (Firts row is row number 0)

Table used to encode the CWs of this row	*X for the left side CW*	*X for the right side CW*
1	(Number of rows -1) \ 3	Number of data columns - 1

2	(Security level x 3) + (Number of rows -1) MOD 3	(Number of rows -1) \ 3
3	Number of data columns - 1	(Security level x 3) + (Number of rows -1) MOD 3

Errors detection and correction.

- Error detection system use 2 CWs and correction system use some between 2 and 510
- The correction system is based on "Reed Solomon" codes which enjoy the math students and terrify others ...
- The number of CWs to add depend of the correction level used, because of the limit to 928 CWs in a bar code (1 of which for the sum of CWs) the maximum level is limited by the number of data CWs. The number of CWs that the error correction algorithm can reconstitute is equal to the number of CWs required by the correction system: this is not magic but it's mathematical !

Level	*Number of CWs required by the correction system, 2 of which for the detection ($2^{level + 1}$)*	*Maximum number of data CWs*
0	2	925
1	4	923
2	8	919
3	16	911
4	32	895
5	64	863
6	128	799
7	256	671
8	512	415

- The advisable correction level depend on the number of data CWs :

Number of data CWs	*Advisable level*
1 a 40	2
41 a 160	3
161 a 320	4
321 a 863	5

- Reed Solomon codes are based on a polynomial equation where x power is 2^{s+1} with s = error correction level used. For sample with the level 1 we use an equation like this: $a + bx + cx^2 + dx^3 + x^4$ The numbers a, b, c and d are the factors of the polynomial equation.
- For information the equation is : $(x - 3)(x - 3^2)(x - 3^3).....(x - 3^k)$ (with $k = 2^{s+1}$) We develop the polynomial equation and we apply a MOD 929 on each factor. These factors have been pre-computed for the 8 equations corresponding to the 8 correction levels. You can see the factors file.
- Rather than to draw the algorithm used to compute the correction CWs, I prefer to provide it to you in Basic.

  Let s the correction level used, $k = 2^{s+1}$ the number of correction CWs, a the factors array, m the number of data CWs, d the data CWs array and c the correction CWs array. We'll use a temporary variable t.

  c and t are inited with 0. And let's go with the math fiddle :

```
For i = 0 To m - 1
  t = (d(i) + c(k - 1)) Mod 929
```

```
    For j = k - 1 To 0 Step -1
      If j = 0 Then
        c(j) = (929 - (t * a(j)) Mod 929) Mod 929
      Else
        c(j) = (c(j - 1) + 929 - (t * a(j)) Mod 929) Mod 929
      End If
    Next
  Next
  For j = 0 To k - 1
    If c(j) <> 0 Then c(j) = 929 - c(j)
  Next
```

Those which have read and understand the Reed Solomon codes will find there ; for the dazed others who have not understand everything (And me !) it's enough to apply the "recipe" by using the obtained codes in the reverse order (From last to first).

Bar code making.

Since we can create the bar code pattern it remains us to draw it on the screen and to print it on a paper sheet. Two approaches are possible:

- The graphic method where each bar is "drawn" as a full rectangle. This method enables to compute the width of each bar with a one pixel precision and so we can work with widths which are perfect multiple of the used device pixel. This gives us a good accuracy specially if the device have a low density as it's the case with screens and inkjet printers. This method requires special programming routines and does not allow making bar codes with current software.
- The special font in which each character is replaced by a bar code. This method allows the use of any software like a text processing or a spreadsheet. (For example OpenOffice, the free clone of MSoffice !) The scale settings according to the selected size can bring away small misshaping of the bar drawing. With a laser printer there's no probs.

It seems that there's no free bar code PDF417 font on the net. I've decided consequently to draw this font and to propose it for download. Because there are 929 CWs with 3 alternatives for each one obligate us to divide the 17 bits of a CW in several parts. But divide by 17 ... hmmm ... a trick is necessary. If we consider that the first bit is always "1" and the last one "0", we can imagine a separator like "01" and rest is only 15 bits in each CW; we can then divide these 15 bits into 3 parts. There will be then $2^5 = 32$ possible groups affected to 32 characters of the font. The encoding software is in charge to transform the 3 groups of each CW into a 3 characters string.
The font will contain thus the following characters:

- Character A (ASCII : 65) to F (ASCII :70) and a (ASCII : 97) to z (ASCII :122) for the 32 groups having a formula "00000" to "11111"
- Character * (ASCII : 42) for the separator having a formula "01"
- Character + (ASCII : 43) for the row start character without its last 0, formula "11111111 0 1 0 1 0 1 00"
- Character - (ASCII : 45) for the end row character without its first 1, formula "111111 0 1 000 1 0 1 00 1"

The string to send to the printer will look something like this : +*gfA*jhD*BAl*gCt*hjk*- and this for each row.

The "pdf417.ttf" font

This font contains the 35 patterns describe above. The start row and end row codes embed a 2 modules margin. The height is equal to 3 modules which is the most current size.

Encoding a pdf417 bar code

The software will evolve with 4 steps :

- Compaction of the data into the CWs by using of the different methods and trying to optimize.
- Calculation of the correction CWs according to the selected security level.
- Splitting by rows and addition of left side and right side CWs.
- Transformation of each CW into a 3 characters string and addition of separators, start row character, end row character and carriage returns.

Because of the interaction between the different compaction modes and the different sub-modes of the "text" mode it's difficult to make a 100% optimization. Thus the software will split the string into "parts" having the type "numeric", "text" or "byte" afterwards it will change some parts for another mode if the overload due to switch CWs is greater than the compaction gain. We can't make allowance for all the parameters like padding, gain due to perfect multiple of 6, uni-character switch or the switches between the different sub-modes of the text mode: one would need for that several thousands of programming lines.
Another difficulty comes from the size of the wielded numbers, for example the "Modulo 900" operation applied on a 44 digits number generate an unavoidable overload error; the software will have to carry out this calculation step by step.