Lors de cette étude, je vous propose de découvrir ce qu'est un **QR-Code**. L'implémentation d'un codeur ou d'un décodeur de **QR-Code** se trouve dans le cours **QtQuick-Multimédia**.

Définition

Le code **QR** (en anglais QR Code) est un type de code-barres en deux dimensions (ou code matriciel) constitué de modules noirs disposés dans un carré à fond blanc. L'agencement de ces points définit l'information que contient le code (ci-contre le **QR-Code** de Wikipedia)

QR (abréviation de l'anglais Quick Response) signifie que le contenu du code peut être décodé rapidement après avoir été lu par un lecteur de code-barres, un téléphone mobile, un smartphone, ou encore une webcam. Son avantage est de pouvoir stocker plus d'informations qu'un code à barres, et surtout des données directement reconnues par des applications, permettant ainsi de déclencher facilement des actions.

Un QR Code est donc un code à réponse rapide. Il a été créé pour gagner su temps et surtout réduire les erreurs de saisie manuelle.

Le QR Code est une évolution du code-barres à 1 dimension.

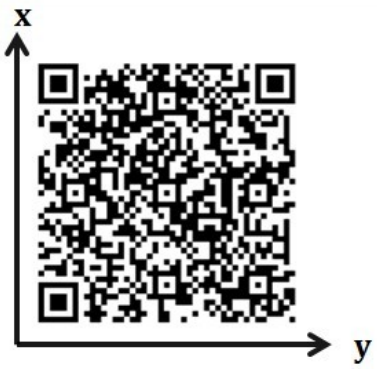
Retours sur le code-barres

Les code-barres unidimensionnels ont été brevetés en 1952 par IBM mais ne permettent de stocker que peu d'informations dans un petit espace. L'usage actuel des code-barres à 1 dimension permet en général de stocker au maximum une vingtaine de caractères sur une petite surface.

En effet, dans un code-barres unidimensionnel, il n'existe aucune redondance d'informations donc il devient vite illisible s'il est abîmé, déchiré ou sale. De plus, il existe un problème de stockage car toutes les informations sont sur une seule ligne alors plus il y aura d'informations, plus le code-barres sera grand et difficile à lire. C'est pour cela que ces code-barres ne contiennent généralement qu'un simple numéro d'identification et pas toutes les informations relatives au produit.

Description - Un code-barres 2D

Un **QR-Code** a la capacité de stocker ses informations horizontalement et verticalement. En effet, la lecture se fera sur 2 axes, un **QR-Code** peut être représenté par une matrice(x,y).



Le passage de 1 dimension à 2 dimensions a permis également d'augmenter la capacité de stockage des code-barres tout en réduisant la taille de l'image. Il est ainsi possible de traduire une information de plusieurs dizaines de caractères en un seul QR-Code de 2 cm de côté !

Le principal atout du **QR-Code** est donc sa haute capacité de stockage :

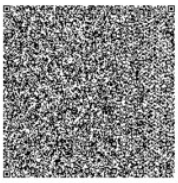
- **7089** caractères **numériques** (0-9)
- **4296** caractères **alphanumériques** (ASCII)
- **2953 octets** (binaires 8-bit)
- **1817 kanji** (caractères japonais)

Le **QR-Code** a l'atout de pouvoir être placé dans des espaces réduits, mais il faut faire attention pour permettre une lecture aisée. En effet, plus il y a de points dans la matrice, plus il y a de l'information stockée dans le **QR-Code**. Il faut donc faire un compromis entre la taille et la lisibilité du **QR-Code** sous peine d'avoir des difficultés à le décoder.

Le Qr-Code a également la capacité de pouvoir être lu dans tous les sens (360°) de manière très rapide grâce à des repères dans la matrice facilement détectables.

Enfin, l'atout majeur du **QR-Code** comparé au **code-barres** traditionnel reste sa résistance à presque tout. En effet, le **QR-Code** tolère d'être abîmé (déchirures, salissures, etc.) car ce dernier contient un code correcteur et de redondance de l'information. En résumé, la force du **QR-Code** est d'intégrer un taux de correction d'erreur qui définit la quantité d'informations répétées, et donc une tolérance plus ou moins importante aux dégradations. Plus le taux de correction d'erreurs est élevé, plus le **QR-Code** sera complexe à la lecture.

Il existe différentes versions de **QR-Code** allant de **25** à **4 296** caractères :

Version 1, 21×21, 10-25 caractères	Version 2, 25×25, 20-47 caractères	Version 3, 29×29, 35-77 caractères	Version 4, 33×33, 67-114 caractères	Version 10, 57×57, 174 à 395 caractères.	Version 40, 177×177, 1 852 à 4 296caractères.
					

Comparaison avec plusieurs code-barres à deux dimensions

Il existe de nombreux code-barres à deux dimensions mais seul le **QR-Code** sort du lot avec les meilleures performances. Le **QR-Code** a un haut pouvoir de stockage, prend peu d'espace et **surtout est très rapide** à la lecture.

	PDF417	DATA MATRIX	MAXI CODE	QR Code	Aztec Code
Developer (Country)	Symbol (USA)	CI Matrix (USA)	UPS (USA)	DENSO (Japan)	Hand Held Products (USA)
Code type	Multi-low	Matrix	Matrix	Matrix	Matrix
Data size (Alphanumeric)	1,850	2,355	93	4,296	3,067
Characteristics	High capacity	High capacity Small space	Fast reading	High capacity Small space Fast reading	High capacity
Main market	OA	FA, Medical	Logistics	All industries	Air Line, Railroad
Standard	AIMI ISO	AIMI ISO	AIMI ISO	AIMI ISO JIS	AIMI

Structure du QR-Code

Le **QR-Code** est composé de plusieurs parties distinctes. Certaines sont directement liées aux données contenues dans le symbole (zone d'encodage), tandis que d'autres ont des fonctions précises pour faciliter la lecture du **QR-Code** (motifs fonctionnels ou function patterns). Le symbole est formé de modules, généralement carrés, clairs et foncés (voir un exemple de structure en fausse couleur page suivante).

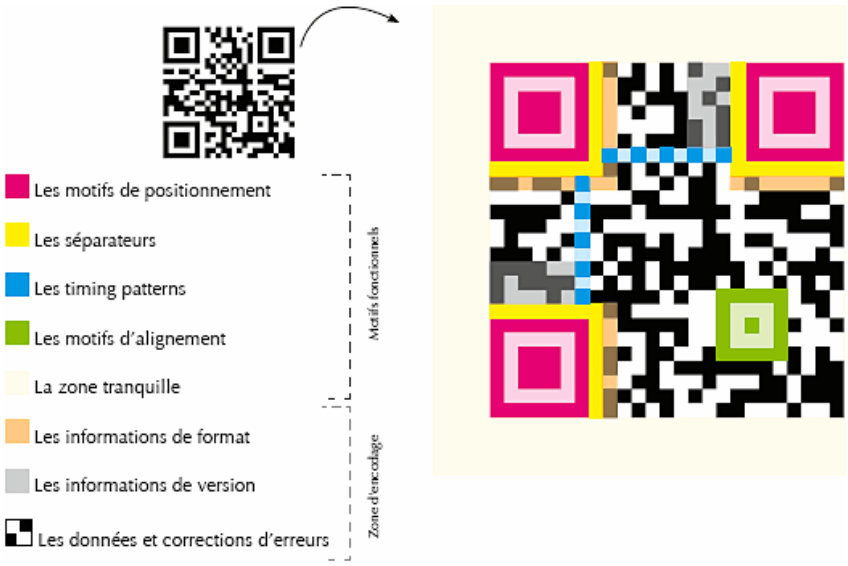
La **zone tranquille (Quiet)** est composée de 4 modules de largeur pour délimiter le **QR-Code**. Cette dernière est non obligatoire mais recommandée. Cette zone est une zone blanche à l'extérieur du **QR-Code** est facilite fortement la lecture du **QR-Code**.

Les **motifs de positionnement** (toujours au nombre de 3 positionnés en haut à gauche, en haut à droite et en bas à gauche) sont très importants et seront détaillés un peu plus tard.

Les **séparateurs** permettent de séparer les patterns de détection de position du reste du code.

Les **timing Patterns** (patterns de synchronisation) permettent de densifier le **QR-Code** (par l'alternance de modules blancs/noirs) de déterminer la version du **QR-Code** et de fournir des positions de référence pour le calcul des coordonnées de chaque module.

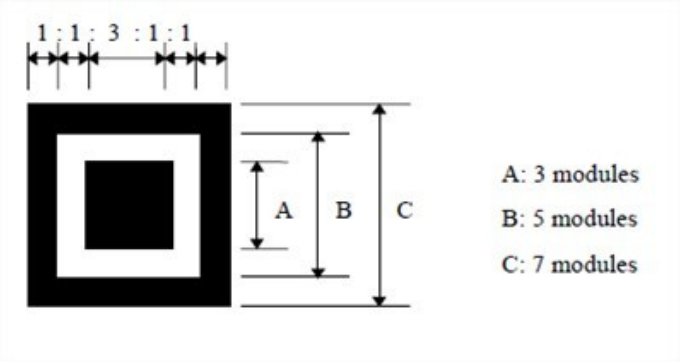
Les **motifs d'alignement** sont composés de 3 carrés concentriques superposés et de taille 5x5 modules pour le premier noir, 3x3 modules pour le blanc et un seul module central noir. Le nombre de ces motif d'alignement dépend de la version de **QR-Code** généré.



La **région du code** contient toutes les données encodées, avec le code de correction d'erreur, les informations de version et de format.

Détection de position

Les patterns de détection de position sont très ressemblants aux patterns d'alignement, à la différence près qu'ils sont plus gros et qu'ils sont toujours présents au nombre de 3 dans le motif du **QR-Code**. Comme nous pouvons le voir dans la figure suivante, ils sont composés d'un premier carré noir de taille 7x7 modules, d'un second carré blanc de 5x5 modules et enfin d'un carré noir central de 3x3 modules. Ce symbole est codé de telle sorte qu'il existe une très faible probabilité de retrouver ce symbole dans tout le symbole du **QR-Code**.



Principales régions

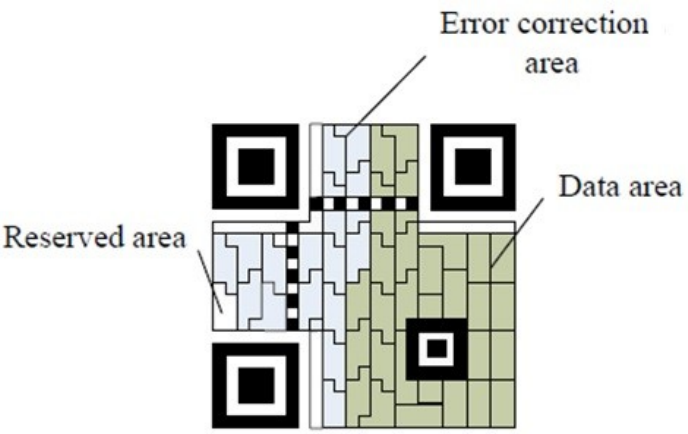
Il existe dans le code 2 principales régions. Le code contenant les données à stocker et le code correcteur. Le code correcteur permettra, en cas de perte des données stockées, à corriger celles-ci et récupérer les informations détruites.

Selon la quantité de données stockées, le **QR-Code** sera plus ou moins complexe.

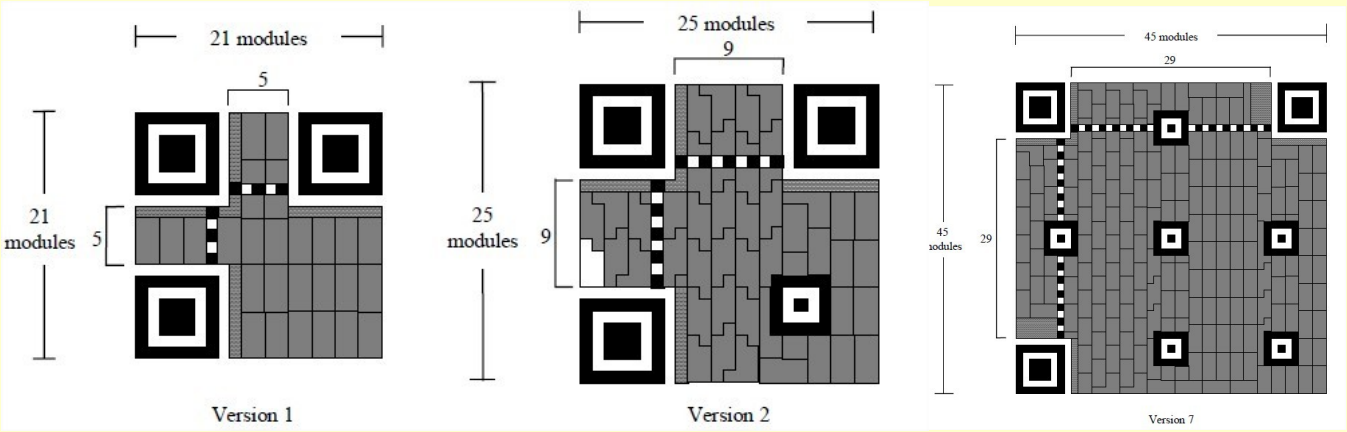
Ainsi, il existe 40 versions de **QR-Code** différentes définies dans les spécifications.

Chacune de ces versions ont une capacité de stockage maximale et telle ou telle version sera utilisée en fonction de la quantité d'informations que l'on veut stocker dans le **QR-Code**.

Toutefois, il est important de noter que plus un **QR-Code** a une version élevée, plus il sera difficile à décoder. Ainsi, pour permettre une meilleur lecture, le nombre de patterns de détection de position augmentera en fonction de la version utilisée.



De plus, on remarquera que pour un **QR-Code** très simple (version 1), il n'y a pas besoin de pattern d'alignement.



Néanmoins, il faut savoir que tout ce qui fait la force du QR-Code vient du fait de l'utilisation du code correcteur et de la redondance d'information.

Redondance des informations

Le principe du **QR-Code**, qui en fait son principal atout, est d'utiliser la redondance d'informations. En effet, la redondance est employée dans le **QR-Code** de manière à prévenir toute altération du motif et ainsi fournir au lecteur l'information codée originale sans problèmes. C'est le code correcteur d'erreurs du **QR-Code** qui créé la redondance afin d'accroître la fiabilité de l'information.

L'objectif d'un code correcteur est la détection ou la correction d'erreurs après la transmission d'un message. Cette correction est permise grâce à l'ajout d'informations redondantes. Le message est plongé dans un ensemble plus grand, la différence de taille contient la redondance, l'image du message par le plongement est transmise. En cas d'altération du message, la redondance est conçue pour détecter ou corriger les erreurs.

Le **QR-Code** a donc cette capacité de fournir l'information même s'il est abîmé. Et ce, grâce au code correcteur particulier qu'il contient : **Le code de Reed-Solomon**.

Le code Reed-Solomon

Le code de **Reed-Solomon** est un code correcteur, dit « code parfait » inventé par les mathématiciens Irving S.Reed et Gustave Solomon, qui est très utilisé et permet de corriger des erreurs dues à la transition imparfaite d'un message.

Ce code de Reed-Solomon est basé sur un principe mathématique qui a pour objectif de construire un polynôme à partir des symboles à transmettre (information source) et de le sur-échantillonner, c'est à dire d'apporter beaucoup plus d'informations à ce polynôme. Le résultat est alors envoyé, au lieu des symboles originaux. La redondance de ce sur-échantillonnage permet au récepteur du message encodé de reconstruire le polynôme source même s'il y a eu des erreurs pendant la transmission.

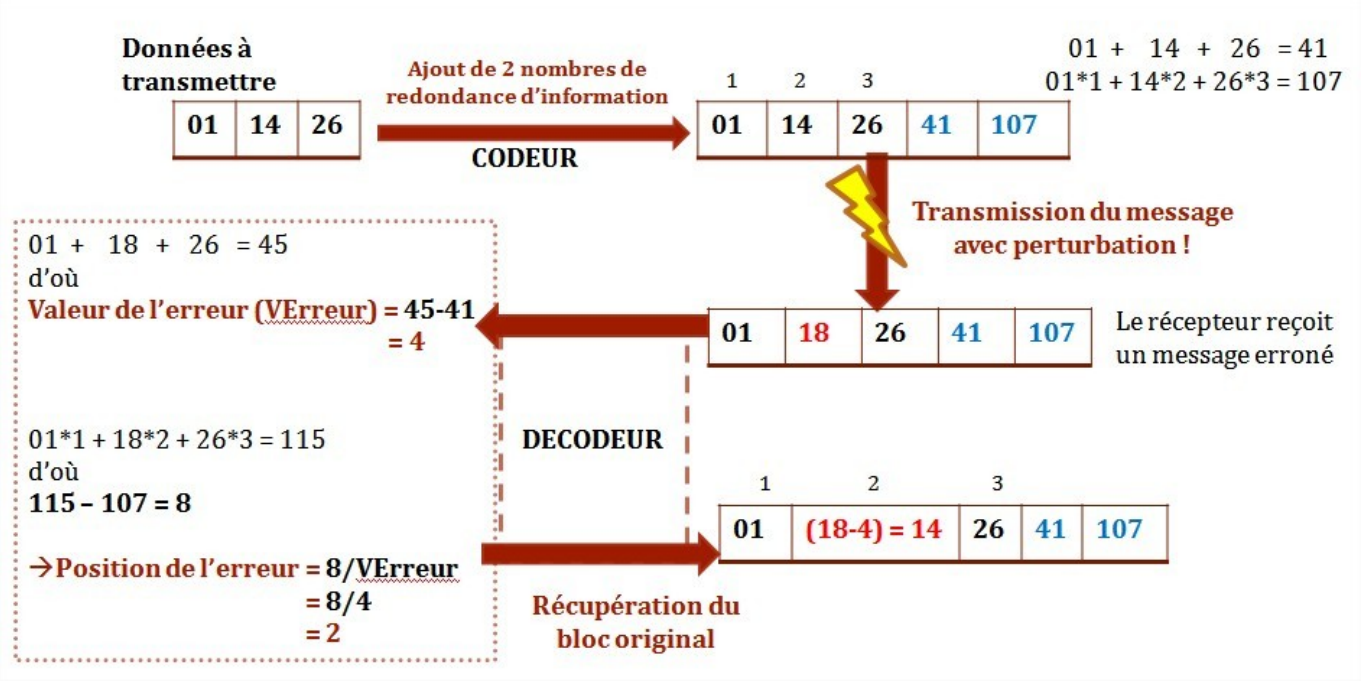
Pour illustrer le fonctionnement du code correcteur, voici un petit exemple très simplifié qui illustre le processus de reconnaissance et de correction d'erreur. Le codeur ajoute à l'information source (les trois nombres) des nombres supplémentaires de redondance d'information (voir page suivante)

Le premier nombre de redondance correspond à la somme des 3 nombres. Le second nombre Si le message transmis est altéré, c'est au rôle du décodeur de détecter l'erreur et de la corriger. Il va s'aider des nombres de redondance d'information ajoutés par le codeur.

Pour le décodage, un calcul de la somme des nombres doit permettre de retomber sur le même premier nombre de redondance d'information. Or ici, le décodeur trouve 45 au lieu de 41. Il sait donc détecter qu'il y a eu une erreur lors de la transmission du message. Il note la différence entre le nouveau nombre trouvé et celui qu'il a en mémoire.

Cette différence trouvée correspond à la valeur de l'erreur. Maintenant, pour corriger l'erreur, il doit trouver où se situe celle-ci parmi les informations reçues. Pour le second nombre de redondance, il trouve une différence. Cette différence divisée par la valeur de l'erreur donnera la position de l'erreur. Il faut donc retirer 4 au nombre du rang 2.

Le décodeur a donc pu détecter et corriger l'erreur simplement pour fournir le bloc original, et ce malgré l'altération du message. Il faut noter que lors d'une transmission sans perturbation, les différences des sommes simples et des sommes pondérées sont nulles.de redondance est la somme pondérée des 3 nombres, chacun est multiplié par son rang.



Génération d'un QR-Code

Après avoir vu tous les principes qui font du **QR-Code** un code-barres particulier et intéressant, attardons-nous maintenant sur la question de savoir comment générer un **QR-Code**.

Il faut rappeler tout d'abord que lors de la génération du QR-Code, nous devons décider du taux de correction d'erreur à appliquer au QR-Code. En effet, un QR-Code peut être généré avec 4 niveaux de taux de correction d'erreur différents selon les spécifications et qui permettent d'obtenir une certaine tolérance quant à la perte d'informations exprimée en pourcentage.

L : LOW, permet une tolérance jusqu'à 7% de perte de code.

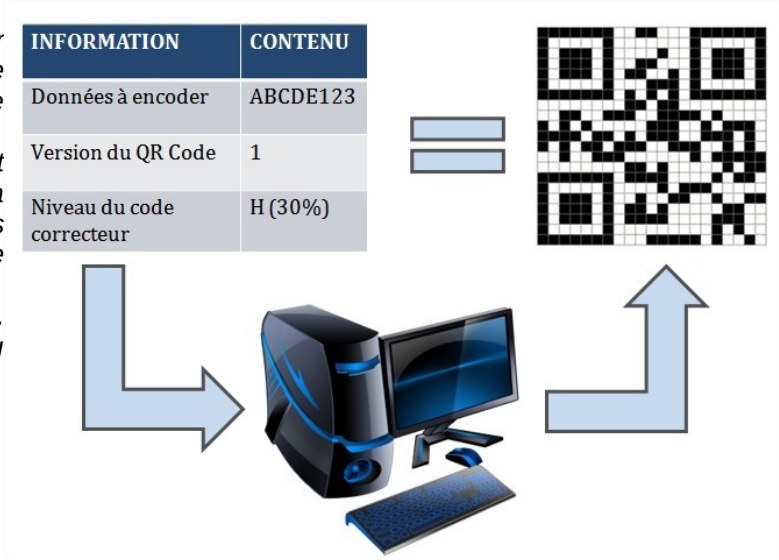
M : MEDIUM, permet une tolérance jusqu'à 15%.

Q : QUALITY, permet une tolérance jusqu'à 25%.

H : HIGH, permet une tolérance jusqu'à 30%.

Déroulons l'algorithme d'encodage d'un QR Code :

1. *Analyse des données à encoder et paramétrage du niveau de code correcteur. Le but est d'analyser le flux de données d'entrée pour identifier la variété des caractères différents pour être encodés. Si l'utilisateur n'a pas spécifié le niveau de code correcteur, la plus petite version de **QR-Code** sera sélectionnée pour accueillir les données.*
2. *Convertir les caractères de données dans un flux de bytes.*
3. *Implémenter la correction des erreurs. Le but est de séparer par blocs les bits de données et de générer leurs codes correcteurs. Comme vu pour le code de **Reed-Solomon**, nous plaçons un bit d'information et autour, nous générons un code correcteur.*
4. *Insérer les données avec le code correcteur dans la matrice. Nous utilisons ici le masque de patterns (Timing pattern, pattern de détection, pattern d'alignement).*
5. *Générer la matrice et évaluer le résultat qu'elle retourne. Nous optimisons ici la balance entre les modules noirs et les modules blancs et on minimise l'occurrence de patterns indésirables*
6. *Générer le **QR-Code** au format image. C'est le résultat final qui pourra être lu par un lecteur de **QR-Code**.*

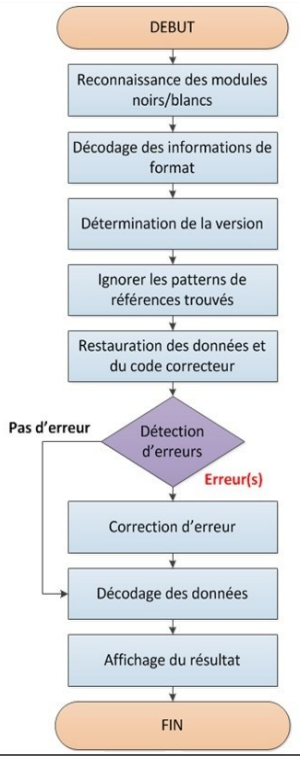


Lecture du Qr-Code

C'est bien beau d'avoir créer un **QR-Code** mais encore faut-il savoir comment le lire.

Déroulons donc l'algorithme de lecture :

1. *Reconnaître les bits 1 ou 0. Le but est de différencier les modules noirs des modules blancs.*
2. *Identifier le taux de code correcteur.*
3. *Identifier la version du **QR-Code**.*
4. *Découvrir la région à décoder.*
5. *Lire les données et le code correcteur.*
6. *Détecter/Corriger les erreurs.*
7. *Décoder les données.*
8. *Afficher le résultat.*



Toutes ces étapes de lecture se feront pour la plupart grâce à la détection des patterns du QR-Code et à leur lecture. Voici ci-contre le diagramme de flux de la lecture :

Il faut noter que pour une meilleure lecture, le QR-Code doit contenir des modules qui ont un fort contraste (noir/blanc). Le QR-Code peut être de plusieurs couleurs mais dans ce cas, il faut que les couleurs entre les modules représentant les modules blancs et ceux représentant les modules noirs puissent être suffisamment contrastées.

Élaboration d'un QR-Code

Après tous ces aspects théoriques, je vous propose de voir comment se répartissent toutes les informations nécessaires au codage à l'intérieur du **QR-Code**. À titre d'exemple, je vous propose d'encoder le message ***« BTS-SN-IR »***. La génération du **QR-Code** se réalise en plusieurs étapes :

1. *Choisir le type d'information à encoder an corrélation avec la version du **QR-Code** qui doit être la plus petite possible, mais qui permet malgré tout d'intégrer le message souhaité en corrélation également avec le taux de redondance choisi.*
2. *Créer une chaîne de bits de données à partir du message que nous désirons encoder. Cette chaîne inclut les caractères du message d'origine ainsi que quelques bits d'information supplémentaires qui précisent au décodeur **QR** de quel type de code **QR** il s'agit.*
3. Générer le code correcteur d'erreur (redondance) en utilisant le principe de **Reed-Solomon**.
4. *Choisir la meilleure forme de masque afin d'optimiser la lecture du **QR-Code**.*

Les différents types de données

Il existe quatre types de données :

- ***Données numériques (chiffres de 0 à 9)***
- ***Données alphanumériques (chiffres de 0 à 9 + caractères de A à Z + 9 caractères espace $ % * + - . / :)***
- ***Données codées sur 8 bits (tous les caractères latins + de nombreux caractères)***
- ***kanji (utilisé pour les langues asiatiques notamment)***

Chaîne de Bits	Type d'information
0001	Numérique
0010	Alphanumérique (Majuscule)
0100	Binaire (Caractères ASCII étendus ISO-8859-1)
1000	Kanji (caractère Japonais)

Les données alphanumériques ne prennent pas en compte les caractères minuscules, mais par contre permettent d'encoder les informations sur 6 bits. En couplant deux lettres majuscules ensembles, nous pouvons même avoir un encodage sur 11 bits (au lieu de 12).

Si vous désirez utiliser les caractères minuscules, il faut cette fois-ci utilisées des données codées sur 8 bits (le type dit Binaire).

Vous remarquez par cette analyse que selon le type de données à encoder, les capacités de stockage nécessaires sont plus ou moins importantes pour le même nombre de caractères.

Version du QR-Code

*Pour encoder notre message, nous devons sélectionner la matrice qui va contenir le message avec l'ensemble des informations supplémentaires. Le message « **BTS-SN-IR** » comporte au total 9 caractères alphanumériques ce qui nous permet de choisir la plus petite version, avec un niveau de correction maximum :*

Version	Modules	ECC Level	Data bits (mixed)	Numeric	Alphanumeric	Binary	Kanji
1	21x21	L	152	41	25	17	10
		M	128	34	20	14	8
		Q	104	27	16	11	7
		H	72	17	10	7	4
2	25x25	L	272	77	47	32	20
		M	224	63	38	26	16
		Q	176	48	29	20	12
		H	128	34	20	14	8
3	29x29	L	440	127	77	53	32
		M	352	101	61	42	26
		Q	272	77	47	32	20
		H	208	58	35	24	15

Encodages Préliminaires

Avant d'encoder notre information, nous devons tout d'abord préciser le type de données, et suivant ce type, nous devons regrouper les caractères (ou les chiffres) du message de manières différentes et les encoder sur un nombre de bits également différents :

- ***Numérique** : regroupement de 3 caractères encodés sur 10 bits. Si le nombre de caractères à encoder n'est pas un multiple de 3, le dernier chiffre est laissé seul et encodé sur 4 bits ou regroupés par 2 et encodés sur 7 bits.*
- ***Alphanumérique** : regroupement de 2 caractères encodés sur 11 bits. Si le nombre de caractères à encoder n'est pas un multiple de 2, le dernier caractère est laissé seul et codé sur 6 bits.*
- ***Mode codé sur 8 bits** : chaque valeur de caractère est simplement codée sur 8 bits.*

Dans notre exemple, le message à encoder comporte des caractères alphanumériques avec des lettres uniquement en majuscule. L'expression binaire commence donc par « 0010 ».

Nous devons ensuite spécifier la longueur du message à encoder dont le nombre de bits dépend de la version du QR-Code et du type d'information. Nous possédons 9 caractères dans le message, compte tenu de la version 1 et du type d'informations, l'expression binaire équivalente est «000001001 » (9 bits, voir tableau ci-dessous).

Version du QR Code	Données numériques	Données alphanumériques	Données codées sur 8 bits
1 à 9	10	9	8
10 à 26	12	11	16
27 à 40	14	13	16

Ainsi l'expression binaire de la partie en-tête devient: « 0010 000001001 ».

Encodage des données sources

Pour l'encodage du message, vous devez le diviser par paires de lettres : BT / S- / SN / -I / R.

Pour chaque paire nous prenons la valeur du code de la première lettre et nous multiplions cette valeur par 45. Nous rajoutons ensuite à cette multiplication la valeur du code de la deuxième lettre. La valeur obtenue est par la suite transformée en une expression binaire sur 11 bits.

Attention la valeur du code de chaque caractère ne correspond pas au code ASCII, mais suit la progression du tableau de la page suivante. Grâce à ce code minimaliste, nous utilisons beaucoup moins de bits que le code ASCII, et nous comprenons mieux la multiplication par 45. Avec le code ASCII, il faudrait 2 fois 8 bits pour les deux lettres :

Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value
0	0	6	6	C	12	I	18	O	24	U	30	SP	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	-	41		

Pour la lettre qui est toute seule, nous prenons aussi la valeur du code de celle-ci que nous transformons cette fois-ci en une chaîne binaire de 6 bits.

BT	S-	SN	-I	R
(11x45)+29	(28x45)+41	(28x45)+17	(41x45)+18	27
524	1301	1277	1863	27
010 0000 1100	101 0001 0101	100 1111 1101	111 0100 0111	01 1011

L'expression binaire globale est donc après l'ensemble de ces opérations :

0010 000001001 01000001100 10100010101 100111111001 11101000111 011011.

Après chaque expression binaire, vous devez rajouter un marqueur de fin, codé sur quatre octets « 0000 ».

Les données binaires doivent être découpées par octets, soit 8 à 8. Ces chaînes de 8 bits sont appelées « codewords ». Ces « codewords » doivent être remplis, vous devez alors ajouter des 0 à la dernière chaîne si nécessaire afin de compléter l'octet.

Version	Taille	Modules total	Modules attribués aux motifs fonctionnels	Modules restants pour la zone d'encodage	Modules attribués aux information de format et de version	Niveau correction d'erreurs	Modules attribués corrections d'erreurs	Nombre de modules restants pour les données
1	21 x 21	441	202	239	31	L	56	152
						M	80	128
						Q	104	104
						H	136	72
2	25 x 25	625	235	390	31	L	87	272
						M	135	224
						Q	183	176
						H	231	128
...								

Le QR-Code que nous désirons fabriquer est de version 1, faite pour coder 104 bits, soit 13 octets (104/8). Or notre expression binaire ne fait que 9 octets, donc il nous faut ajouter encore 4 octets.

Il existe pour cela deux octets spécifiques *11101100* (Code ASCII 236) et *00010001* (Code ASCII 17) que nous rajoutons alternativement jusqu'à ce que nous complétions les 104 bits requis. Notre chaîne binaire, avant de subir la redondance, est finalement la suivante :

00100000 01001010 00001100 10100010 10110011 11110011 11010001 11011011 00000000 11101100 00010001 11101100 00010001

Édition du code de correction d'erreurs

Toutes les données sont maintenant encodées et aptes à être décodées. Cependant, avant de les placer dans la matrice, vous devez éditer les codes de correction d'erreurs. En effet, l'un des points forts du QR-Code est son système de correction d'erreurs fondé sur le code de Reed-Solomon.

Il permet une bonne lecture du QR Code même si celui-ci est endommagé (rayure, élément graphique positionné dessus, page pliée, etc.). Même s'il ne peut pas décrypter toutes les données, le lecteur de QR-Code peut néanmoins déduire ces informations grâce aux modules destinés au code de correction d'erreurs.

Le calcul de ces corrections d'erreurs s'appuie sur les corps de galois. Il s'agit de calculs assez complexes dont je ne ferais pas la démonstration ici. Cependant, pour en comprendre le principe, voici la manière dont ce code pourrait être généré.

Prenons le deuxième « codewords » de notre exemple soit : 01011011. Pour retrouver les informations si la zone d'encodage de ces données est illisible, nous transmettons :

un premier nombre qui est la somme de toute la chaîne : 0 + 1 + 0 + 1 + 1 + 0 + 1 + 1 = 5

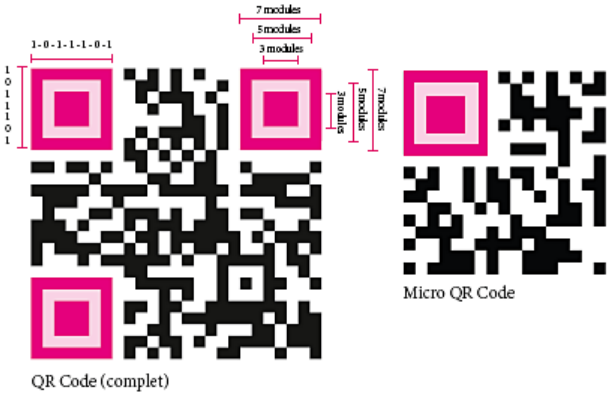
un second nombre qui est la somme pondérée de toute la chaîne, chacun étant multiplié par son rang :

0x1 + 1x2 + 0x3 + 1x4 + 1x5 + 0x6 + 1x7 + 1x8 = 26

Le bloc de correction d'erreurs correspondant au bloc de données 1 contient 526 (sous forme de données binaires). Il peut ainsi corriger les données si elles ont été mal reçues ou bien les déduire.

Placement des éléments

Avant de placer les données, nous devons tout d'abord placer les éléments obligatoires.



Motifs de positionnement (finder patterns) : un QR-Code contient trois motifs de positionnement identiques disposés aux deux coins supérieurs et en bas à gauche. Ils s'agit de trois carrés superposés formés par des modules. Le plus grand de 7 x 7 modules en noir, le second de 5 x 5 modules en blanc et le dernier de 3 x 3 modules en noir. Ils occupent donc 147 modules du symbole quelque soit sa version. Ces motifs de positionnement permettent au lecteur de définir rapidement l'emplacement du QR-Code ainsi que son orientation. Par conséquent, un QR-Code peut être placé ou lu dans n'importe quel sens.

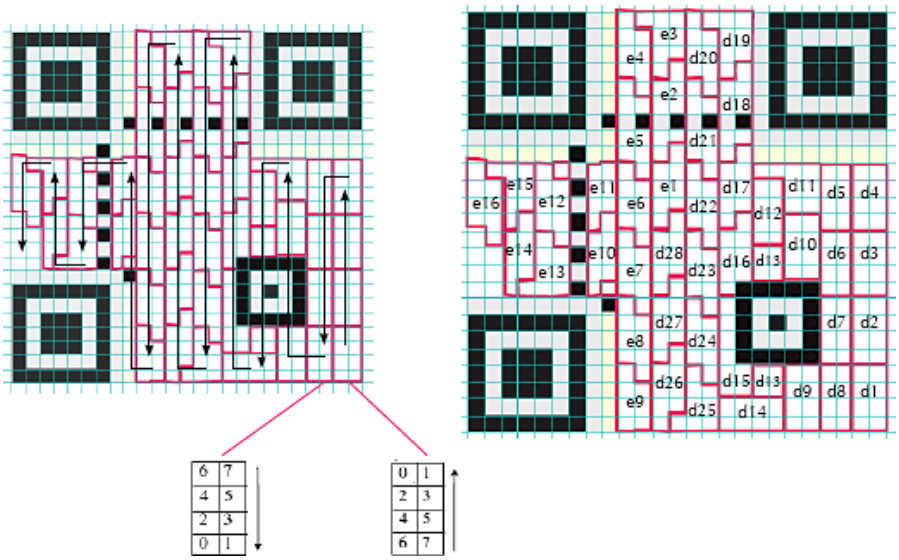
Séparateurs : constitués de modules blancs uniquement, ils servent à distinguer les motifs de positionnement de la zone d'encodage en formant un filet blanc de la largeur d'un module.

Timing patterns : ils sont composés de modules blancs et noirs en alternance avec toujours le premier et le dernier module en noir. Il y a deux timing patterns : un horizontal (en haut à partir de la colonne 6) et un vertical (à gauche à partir de la ligne 6). Ils permettent au lecteur de percevoir le contraste entre les modules clairs et les modules foncés. C'est notamment grâce à eux qu'il est possible de changer les couleurs d'un QR Code (en combinaison avec la zone tranquille). De même, ils permettent au lecteur de déterminer la version du QR Code (en relation avec sa taille).

Motifs d'alignement (alignment patterns) : ils sont formés de la même manière que les motifs de positionnement. Cependant, ils sont plus petits (5 x 5 foncés, 3 x 3 clairs, 1 x 1 foncé). Leur nombre et leur emplacement dépendent de la version du QR-Code. Ils servent notamment à lire le QR-Code même si celui-ci est déformé, par exemple sur un objet 3D ou sur une feuille courbée.

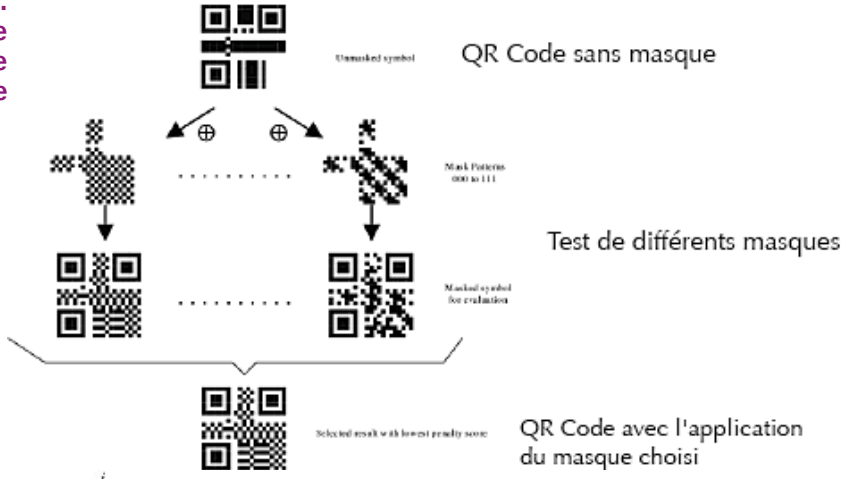
Placement des codewords

L'espace restant doit maintenant contenir toutes les données. Vous avez des blocs avec un sens de lecture précis qui dépend de la version du QR-Code.

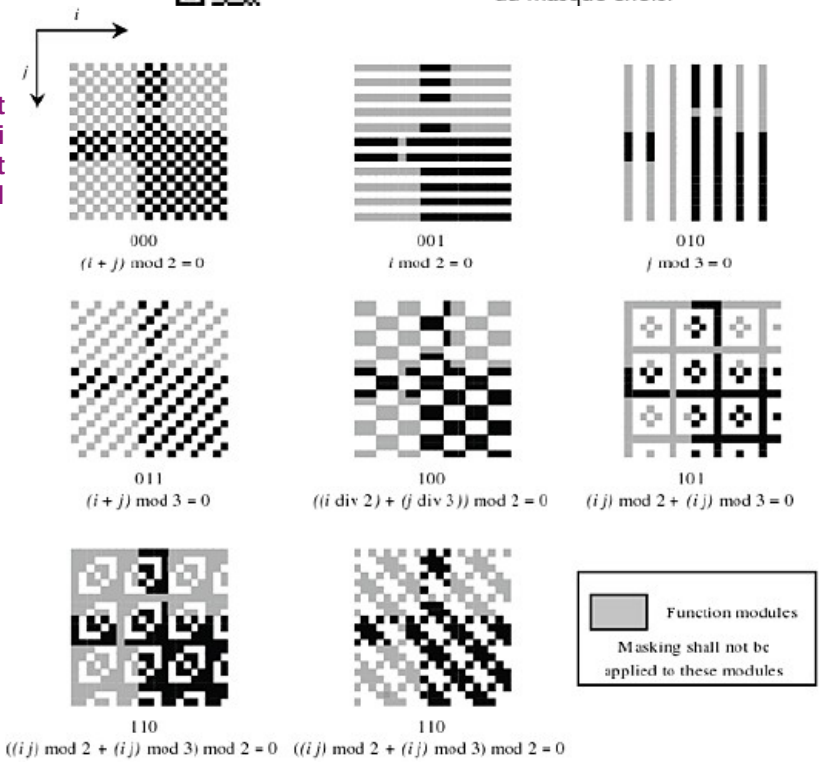


Création du masque

Il s'agit d'appliquer un masque au symbole afin d'éviter de se retrouver avec des zones trop grandes de modules foncés ou clairs les uns à côtés des autres. En effet, cela peut gêner le lecteur qui peut ne pas reconnaître correctement les modules. Après application du masque, le symbole est modifié. Ce masque ne s'applique qu'aux codewords (de données et de correction d'erreurs).

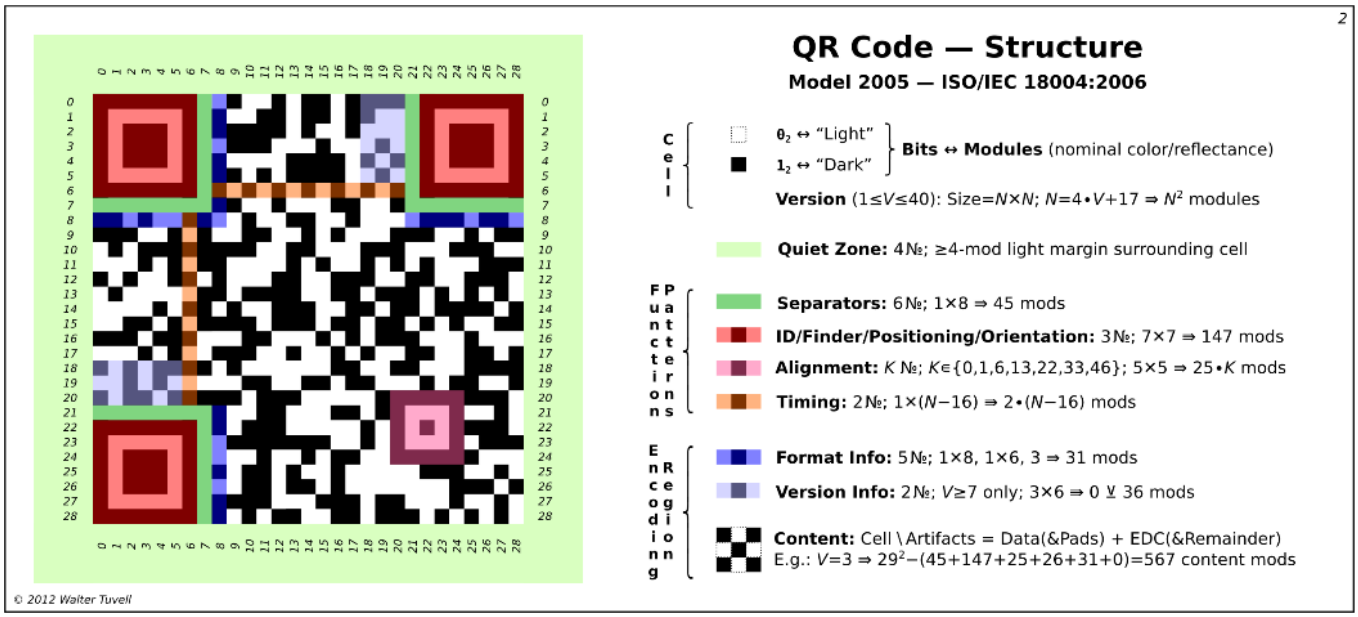


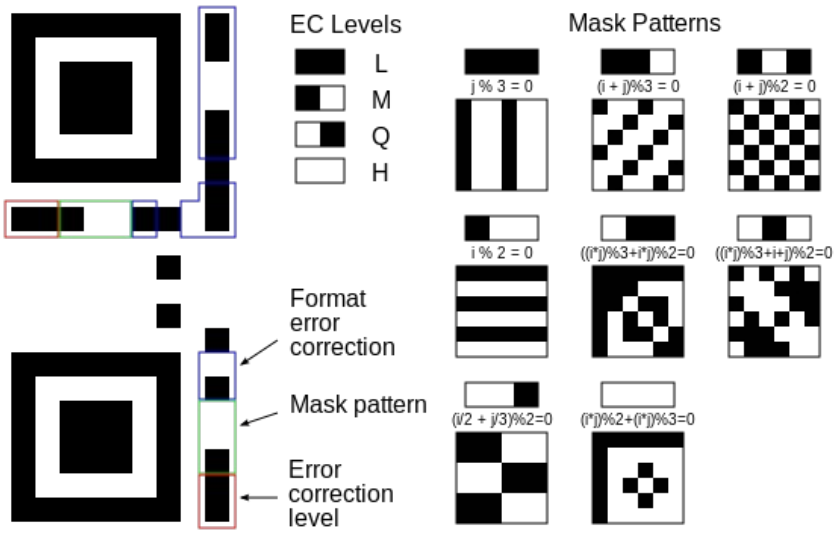
Il existe huit masques différents. Chacun est appliqué au symbole ainsi obtenu et celui qui produit le meilleur résultat est sélectionné de sorte qu'à la sortie un seul est appliqué.

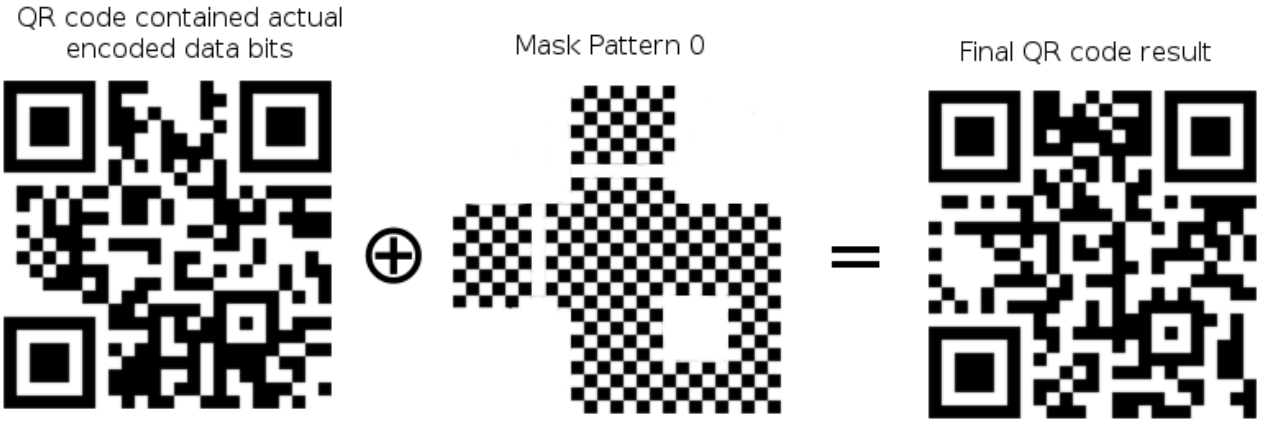


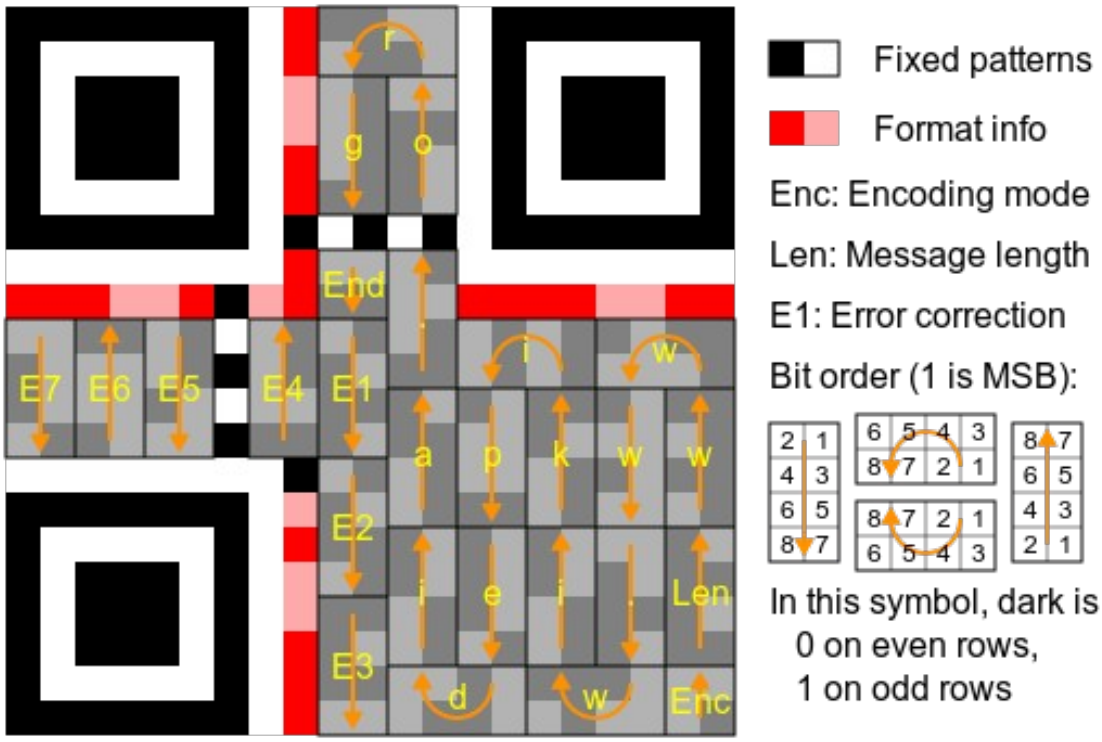
Structure complète d'un QR-Code

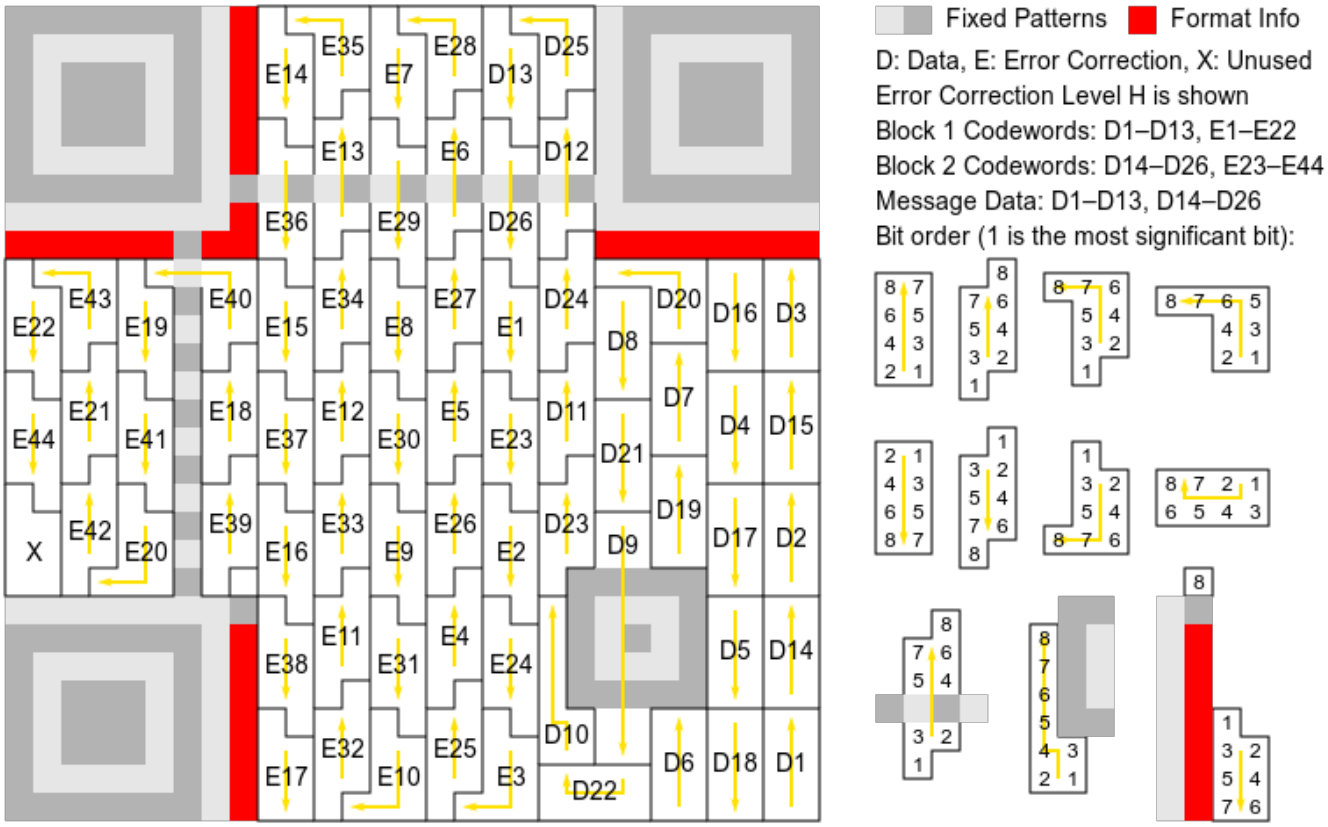
Pour finir et pour bien comprendre l'architecture complexe d'un **QR-Code**, je vous résume l'organisation globale avec l'ordre des éléments à placer au travers des graphismes officiels (en anglais).








Fixed patterns
Format info
Enc: Encoding mode
Len: Message length
E1: Error correction
Bit order (1 is MSB):
In this symbol, dark is
0 on even rows,
1 on odd rows
End
Len
Enc
E1
E2
E3
E4
E5
E6
E7


Fixed Patterns
Format Info
D: Data, E: Error Correction, X: Unused
Error Correction Level H is shown
Block 1 Codewords: D1–D13, E1–E22
Block 2 Codewords: D14–D26, E23–E44
Message Data: D1–D13, D14–D26
Bit order (1 is the most significant bit):

Génération et lecture d'un QrCode sous linux

A fin de valider la conception de cette technologie, je vous propose de l'expérimenter avec le système **Linux**, sachant que ces librairies existent pour les autres systèmes d'exploitation, comme **Windows** au **MacOs**.

Il existe la librairie qrencode qui, comme son nom l'indique, permet de générer un fichier image représentant le QrCode à partir d'un texte, directement en ligne de commande, où vous pouvez spécifier les dimensions de l'image générée et choisir également la qualité de votre QrCode.

Pour la lecture, il existe une autre librairie zbar, qui effectue l'opération inverse et qui permet de décrypter le QrCode donné dans un fichier image afin de retrouver le texte équivalent, quelque soit la qualité choisie. Cette librairie permet également de décrypter les codes barres, les datamatrix, etc. Cette librairie est très performante, puisqu'elle est capable de retrouver n'importe quel code même si votre image possède bien d'autres éléments.

Génération et lecture de QrCode

```
manu@HPE-120fr:~/Travail$ sudo apt install qrencode   // installation de qrencode
manu@HPE-120fr:~/Travail$ sudo apt install zbar-tools // installation de zbar
manu@HPE-120fr:~/Travail$ qrencode
qrencode version 4.0.2
Copyright (C) 2006-2017 Kentaro Fukuchi
Usage: qrencode [OPTION]... [STRING]
Encode input data in a QR Code and save as a PNG or EPS image.

  -h           display this message.
  --help       display the usage of long options.
  -o FILENAME  write image to FILENAME. If '-' is specified, the result
               will be output to standard output. If -S is given, structured
               symbols are written to FILENAME-01.png, FILENAME-02.png, ...
               (suffix is removed from FILENAME, if specified)
  -r FILENAME  read input data from FILENAME.
  -s NUMBER    specify module size in dots (pixels). (default=3)
  -l {LMQH}    specify error correction level from L (lowest) to H (highest).
               (default=L)
  -v NUMBER    specify the minimum version of the symbol. (default=auto)
  -m NUMBER    specify the width of the margins. (default=4 (2 for Micro))
  -d NUMBER    specify the DPI of the generated PNG. (default=72)
  -t {PNG,PNG32,EPS,SVG,XPM,ANSI,ANSI256,ASCII,ASCIIi,UTF8,ANSIUTF8}
               specify the type of the generated image. (default=PNG)
  -S           make structured symbols. Version must be specified.
  -k           assume that the input text contains kanji (shift-jis).
  -c           encode lower-case alphabet characters in 8-bit mode. (default)
  -i           ignore case distinctions and use only upper-case characters.
  -8           encode entire data in 8-bit mode. -k, -c and -i will be ignored.
  -M           encode in a Micro QR Code.
  -V           display the version number and copyrights of the qrencode.
  [STRING]     input data. If it is not specified, data will be taken from
               standard input.
manu@HPE-120fr:~/Travail$ qrencode -o QrCode.png "Bienvenue BTS-SNIR"
manu@HPE-120fr:~/Travail$ qrencode -l H -s 8 -o QrCode.png "Bienvenue BTS-SNIR"

manu@HPE-120fr:~/Travail$ zbarimg QrCode.png
QR-Code:Bienvenue BTS-SNIR
scanned 1 barcode symbols from 1 images in 0.03 seconds
```

```
manu@HPE-120fr:~/Images$ zbarimg code-barre.png
QR-Code:http://LearnOpenCV.com
EAN-13:0012345543210
EAN-13:0123456543218
EAN-13:0001234000057
scanned 4 barcode symbols from 1 images in 0.03 seconds
```