Standard generacji kodu wielowymiarowego

SPIS TREŚCI

1. Wprowadzenie

1.1. Cel i zakres dokumentu

Celem dokumentu jest opis sposobu generacji kodu na potrzeby przechowywania danych.

Dokument obejmuje całościowy zakres zagadnień, związanych z generacja kodu i będzie się składać na: typ kodu i jego charakterystykę oraz model struktury danych. Ponadto w dokumencie zostaną rozwinięte kwestie doboru rozmiaru kodu do żądanej ilości zakodowanych danych.

Definicja standardu nie obejmuje:

- Kwestii transferu danych z urządzenia zewnętrznego (czytniki kodów itp.) do aplikacji. Na rynku jest szeroki wybór urządzeń umożliwiających odczyt kodów; różnią się one m.in. ilością sposobów w jaki odczytane dane można przekazać do systemu, a wynikowo do działającej aplikacji.
- Wykorzystania odczytu kodu przy pomocy API w różnych językach programowania. Zastosowane technologie mogą się w różnym stopniu integrować z systemem operacyjnym, co może uniemożliwiać wykorzystanie niektórych sposobów transferu danych poprzez system operacyjny do aplikacji. Z powyższych powodów nie sprecyzowano sposobu transferu danych z urządzenia pozostawiając tą kwestię do decyzji implementującego odczyt.
- Wyboru urządzenia do odczytu kodu, który jest uzależniony od spełnienia podstawowych wymagań opisanych w następnych punktach.

2. Rekomendowane rozwiązania techniczne

2.1. Kryteria wyboru kodu

Wymagania niefunkcjonalne, od których uzależnione jest efektywne wykorzystanie kodu to:

- ✓ Pojemność danych – maksymalna ilość danych do umieszczenia w jednym egzemplarzu kodu oraz ilość danych umieszczonych przy założonym rozmiarze kodu
- ✓ Rozmiar kodu wynikowego – rozmiar fizyczny po uwzględnieniu rozdzielczości urządzenia prezentującego (np. drukującego) oraz urządzenia odczytującego
- ✓ Szybkość oraz stabilność skanowania – czas potrzebny na rozpoznanie kodu po odczycie oraz łatwość odczytu kodu przez czytnik (nieznaczne ruchy ręki, refleksy światła)
- ✓ Dostępność bibliotek generujących – łatwość dostępu do dojrzałych implementacji (również open-source)

2.2. Przetestowane rozwiązania

Pod kątem wymagań niefunkcjonalnych przetestowano kilka rodzajów kodów. Kody 1D nie były brane do testów ze względu na znikomą pojemność danych. Dostępne kody 3D (trzeci wymiar definiowany przez kolor), nie były testowane ze względu na fazę rozwojową sprzętu i bibliotek obsługujących ten typ kodu. Dostępność czytników, które posiadają funkcjonalność odczytu kodów 1D, 2D i 3D jest ograniczona, a przez to ich cena jest nieproporcjonalna do korzyści płynących ze zwiększenia pojemności danych.

Wyniki przeprowadzonych testów obrazuje tabela:

	PDF-417	DataMatrix	QR Code	Aztec
Typ kodu	1D (piętrowy)	2D (Matrycowy)	2D (Matrycowy)	2D (Matrycowy)
Pojemność danych	Mała	Średnia	Duża	Średnia
Rozmiar kodu	Duży	Mały	Mały	Średni
Szybkość/stabilność odczytu	Średnia	Duża	Znakomita	Duża
Dostępność bibliotek	Tak	Tak	Tak	Tak

2.3. Charakterystyka wybranego kodu

Bazując na wynikach testów wybrano typ kodu QR COde®[1] model 2. Biorąc pod uwagę kryteria szczegółowe dostępne jako dokumentacja projektowa najlepiej spełnia postawione wymagania niefunkcjonalne z przetestowanych.

QR Code® jest kodem o strukturze modułowej. Moduły składają się określonej ilości punktów i są zawsze kwadratowe. Wynikowy kod również jest zawsze kwadratowy.

Rysunek 1: QR Code®

Rysunek 1 przedstawia przykładowy kod. Widać na nim trzy duże kwadraty w rogach – kwadraty pozycjonujące zapewniające stabilny odczyt.

Najmniejszy kwadrat widoczny na rysunku to pojedynczy moduł (najmniejsza składowa kodu – budowana z n x n punktów).

Kod wynikowy jest zbudowany z n x n modułów. Ilość modułów w kodzie definiuje tzw. wersję kodu. Wersja kodu wprost przekłada się na ilość danych możliwych do zakodowania.

QR Code® posiada również wbudowaną korekcję błędów.

Specyfikacja QR Code® definiuje cztery typy kodowania znaków w kodzie: numeryczny, alfanumeryczny, binarny, kanji. Wybrano metoda kodowania znaków przekłada się wprost na ilość znaków, które można zakodować (wiąże się to wprost z rozmiarem wygenerowanego kodu).

Więcej informacji:
http://www.denso-wave.com/qrcode/index-e.html

[1] QR Code® jest zarejestrowaną marką firmy DENSO WAVE Inc.

3. Metodyka generowania kodu 2D

3.1. Parametry generowanego kodu oraz wymagania względem czytnika

Generowany kod powinien posiadać następujące parametry:

- Ustawienie korekcji błędów na poziomie M (15%). Korekcja błędów na tym poziomie jest optymalna dla przekazywania dużej ilości danych w kodzie umieszczonym na papierze. Większy poziom korekcji błędów powiększa całkowitą objętość bajtów narzutowych. Przy powszechnie występujących uszkodzeniach kodów przy obrocie dokumentami (rozmazania tuszu i zgięcia papieru), poziom korekcji błędów rzędu 15% uznajemy za wystarczającą.
- Numeryczny sposób kodowania – uzasadnienie w następnym punkcie
- Rozmiar modułu wynoszący co najmniej 9 x 9 [mil] (0,22 x 0,22 [mm]). Wymiary podyktowane są rozdzielczością odczytu urządzenia zewnętrznego. Dla przetestowanych urządzeń taki wymiar modułu zapewnia stabilny odczyt. Czytnik powinien umieć odczytać QR Code® o wymienionym rozmiarze modułu. W wypadku generacji obrazka z kodem o rozdzielczości 300 DPI (12 DPmm), moduł powinien się składać z 3 x 3 punktów, a 600 DPI (24 DPmm): 6 x 6 punktów itd. W takich wypadkach wymiar modułu wynosi 10 x 10 [mil] co równa się 0,25 x 0.25 [mm].

3.2. Zapis struktury danych w kodzie – własności

Biorąc pod uwagę kryteria wybrano numeryczne kodowanie znaków, z racji możliwości zakodowania większej objętości danych oraz jednoznacznego odczytu zmapowanych na liczby znaków, w tym polskich znaków diakrytycznych. QR Code® nie udostępnia oznakowania typu strony kodowej użytej do zapisu znaków, co w praktyce dla niektórych czytników powoduje to skuteczne odczytanie tylko pierwszych 128 znaków ASCII. Szczegółowe zasady mapowania znaków na liczby znajdują się w następnym punkcie.

Rozważano użycie XML'a jako struktury przenoszącej dane, jednak narzut informacji potrzebnych do napisania znaczników był duży[2]. Przy dużej ilości pozycji wzrost objętości danych powodował wzrost rozmiaru kodu wykraczający poza rozsądnie dobrane miejsce na dokumentach. Wynikowo użyto struktury pól danych oddzielonych separatorem.

Odrzucono możliwość kompresowania danych (użycie ZIP itp.). Dla rozmiaru danych rzędu 1-4 kilobajty oszczędność na objętości danych wahała się w przedziale 20-60%. Pociągało to natomiast konieczność zmiany kodowania z typu numerycznego na binarny co skutkowało kilkakrotnie mniejszą objętością znakową samego kodowania QR Code®[3].

3.3. Zasady mapowania znaków

Ponieważ stosuje się numeryczne kodowanie QR COde®, każdy znak (litera, cyfra, znaki inne niż alfanumeryczne umieszczone na skierowaniu) jest mapowany na liczbę dwucyfrową. Mapowane są znaki z tablicy ASCII od 32 do 126, niektóre znaki sterujące. Ponadto dodano mapowanie polskich znaków diakrytycznych.

[2] Ilość bajtów poświęconych na zapis znaczników oczywiście zależy od ilości znaków w znaczniku oraz samej liczby znaczników Przyjmując że nazwa określająca znacznik jest czytelną nazwą, a nie kodem, a samych pozycji jest dużo, to narzut bajtów mógł wynosić nawet 100% w stosunku do danych właściwych. Kodowanie znaczników XML zmniejszało ilość bajtów narzutowych nawet o połowę, jednak wtedy taka struktura danych traciła na czytelności, a rozmiar kodu i tak powiększał się.

[3] Dla przykładu 20 wersja kodu mieści w sobie ok. 1600 znaków liczbowych, a tylko ok. 600 znaków w binarnym typie kodowania.

Wartość w tablicy ASCII (DEC)	Znak	Znak po zmapowaniu[4]
32 – 96	A-Z, 0-9, znaki inne niż alfanumeryczne	Wartość ASCII bez zmian
97 – 122	a-z	Należy je zmienić na odpowiadające im duże znaki i odpowiednio zmapować.
Brak	Ą	11
Brak	Ć	12
Brak	Ę	13
Brak	Ł	14
Brak	Ń	15
Brak	Ó	16
Brak	Ś	17
Brak	Ż	18
Brak	Ź	19
Brak	Małe polskie znaki diakrytyczne	Należy je zmienić na odpowiadające im duże znaki diakrytyczne i odpowiednio zmapować.
123	{	20
124	|	21
125	}	22
126	~	23
0	NULL	24
9	HT	25
10	LF	26
13	CR	27
Brak	Brak : Shift	97
Brak	Brak : Caps Lock On	98
Brak	Brak : Caps Lock Off	99
Brak	**Brak : Separator pól danych**	**10**

W procesie mapowania, każdą literę należy traktować jako dużą. Do zaznaczenie. Które litery są faktycznie duże służy kombinacja Shift, a także Caps Lock.

W procesie odczytu liczb z kodu i przypisaniu im odpowiednich znaków, każda odczytana litera jest traktowana jak mała chyba, ze występują kombinacje Shift lub Caps Lock.
Shift przed literą oznacza, ze w procesie odczytu będzie ona brana jako duża, a następne będą traktowane jako małe.

Jeżeli występuje ciąg dużych liter, to należy zastosować bardziej wydajny Caps Lock. Umieszczenie kombinacji 98 (Caps Lock On) przed znakiem oznacza, ze do czasu pojawienia się Caps Lock Off, ten znak i każdy następny będą brane jako duże.

Caps Lock Off powinien się pojawić gdy następny znak brany do mapowania jest małą literą. Wystąpienie po dużych literach znaków innych niż małe litery nie powinno owocować pojawieniem się Caps Lock Off.

[4] Zestaw niewykorzystanych kombinacji: 28-31. Kombinacje 00-09 nieużywane by w procesie odmapowania liczb nie pojawiła się przypadkowo sekwencja 10 dedykowana tylko do rozdzielania pól danych.

Przykłady mapowania:

- Mapowanie „Jan Kowal”

ShiftJ	a	n	spacja	ShiftK	o	w	a	l
9774	65	78	32	9775	79	87	65	76

- Mapowanie „JAN Kowal”

CapsOnJ	A	N	spacja	K	CapsOffo	w	a	l
9874	65	78	32	75	9979	87	65	76

- Mapowanie „ĄĘĆ 50 zł”

CapsOnĄ	Ę	Ć	spacja	5	0	spacja	CapsOffz	ł
9811	13	12	32	53	48	32	9990	14

3.4. Model struktury danych

3.4.1. Oznaczenie separatora rozdzielającego pola danych.

Dane przeznaczone do zakodowania umieszcza się w polach danych. Do rozdzielenia pól danych zastosowano separator. Separatorem już zmapowanych na liczby pól danych jest liczba 10. Liczba ta nie służy do mapowania żadnego znaku, więc żaden znak z tabeli z poprzedniego punktu. nie jest zabroniony do użycia w polach danych. Takie podejście wyeliminowało problem z użyciem separatora typu „#” lub „|” – znaki te nie mogły być użyte w polach danych.

Użycie znaków sterujących do separowania pól danych odrzucono ze względu na błędną ich interpretację w niektórych technologiach.

3.4.2. Zasady oznaczania w kodzie 2D ramki z danymi

Ramka powinna zaczynać się od informacji, że kod odczytany przez urządzenie jest kodem 2D z pożądanymi danymi Oznacza się to przez trzy znaki:

@^` (DEC: 64, 94, 96)

Następne dwa znaki XX to typ danych w kodzie (zakres: 00-99). Kolejne dwa znaki NN to wersja danych. Po znakach wersji następują po sobie pola danych rozdzielonych separatorem. Na końcu kodu powinny się znaleźć sekwencja:

@@@ (DEC: 64, 64, 64)

Jeżeli zaistnieje sytuacja, w której do zakodowania danych jest potrzebny więcej niż jeden kod (obszerny tekst, założenie, że wynikowa generacja w kilku małych kodach), to po sekwencji rozpoczynającej kod należy umieścić znaki Exy, gdzie x jest całkowitą liczbą kodów, w których dane zostały zakodowane; y jest to numer aktualnie odczytanego kodu.
Następnie powinna się znaleźć sześciocyfrowa liczba losowa – identyfikator wprowadzony aby uniknąć mylnego połączenia kodów. **Identyfikator ten to jedyna pozycja w kodzie, która nie ulega mapowaniu znaków na dwucyfrowe liczby.**
Same sekwencje rozpoczynające i kończące kody również się różnią. Początkiem pierwszego kodu zostaje sekwencja @^`, natomiast każdy kolejny rozpoczyna się sekwencją:

`^@ (DEC: 96, 94, 64)

Również końcem ostatniego kodu zostaje sekwencja @@@, natomiast każdy inny kończy się sekwencją:
`^@ (DEC: 96, 94, 64)

Poniżej znajduję się przykład[5] wyglądu ramki z danymi przed mapowaniem znaków na liczby. Ponieważ separator nie ma znaku przed mapowaniem na liczby to dla czytelności przykładu oznaczono go przez „**SEP**":

Dane typu 12 wersji 00 w jednym kodzie:
@^`1200Pole danych 1**SEP**Pole danych 2**SEP**.. tutaj dalsze dane …**SEP**Pole danych N@@@

Dane typu 14 w wersji 02 w dwóch kodach o losowych identyfikatorze 723456:
Kod1 : @^`E21723456 1402Pole danych 1**SEP**...tutaj dalsze dane…**SEP**Pole danych N-3`^@
Kod2 : `^@E22723456Pole danych N-2**SEP**Pole danych N-1**SEP**Pole danych N@@@

Jeżeli dane są zawierane w więcej niż jednym kodzie, to należy pamiętać by nie rozdzielać zawartości pojedynczego pola danych.

Jeżeli pole danych ma być puste, to między separatorami nie powinno być żadnego znaku.

3.4.3. Wpływ rozróżnienia wielkości znaków na wygląd ramki

Kombinacja rozdzielająca pola danych nie wpływa na sposób wstawiania oznaczeń „Shift" i „Caps Lock". „Caps Lock On" działa na znaki niezależnie czy między nimi w pewnym momencie wystąpi separator pól danych. Jeżeli zaistnieje sytuacja, w której „Caps Lock On" jest aktywny do końca kodu, to nie ma potrzeby użycia „Caps Lock Off" na końcu kodu.

Jeżeli dane są generowane w liczbie kodów większej niż 1, to należy pamiętać, ze oznaczenia „Shift" i „Caps Lock On" tracą aktywność po zakończeniu danego kodu. W skrajnym przypadku jeżeli celem jest mapowanie znaków danych na same duże litery, to pierwsze pole danych w każdym kodzie będzie rozpoczynać się „Caps Lock On".

Istnieje możliwość dopełniania pól danych znakami NULL (po mapowaniu kombinacja 24). Taka sytuacja może zaistnieć w sytuacji, gdy jest wymagany minimalny rozmiar kodu, a objętość danych jest niewystarczająca. Wypełnienie pustymi znakami może również posłużyć do generacji wielu kodów o takim samym rozmiarze. W procesie odczytu liczb z kodu i przypisaniu im odpowiednich znaków, znak NULL powinien być ignorowany.

3.5. Dodatkowe struktury danych dla pól danych

3.5.1. Lista

Ponieważ niektóre pola danych mogą w sobie zawierać kilka podpozycji, przewidziano dla nich bardziej rozbudowaną strukturę: listę. W przypadku gdy w polach, dla których przewidziano wykorzystanie list, znajduje się tylko jedna pozycja, to należy ją wypełnić podobnie jak zwykłe pole

[5] Kompletny przykład wraz z mapowaniem znaków na liczby w załączniku

(bez użycia znaczników listy). Listę definiuje się podobnie jak jednopoziomową listę w języku HTML[6]:

<di>
ABC
DEF
GHI
</di>

3.5.2. Tabela

Dla niektórych przypadków zastosowanie listy jest niewystarczające do poprawnego odwzorowania struktury danych. Do zapisu tych danych przewidziano tabelę. W przeciwieństwie do listy, znaczniki tabeli należy umieścić nawet wtedy gdy tabela posiada tylko jedną pozycję. Tabelę definiuje się podobnie jak w HTML. Podobnie jak w HTML przewidziano również zastosowanie znaczników do określenia nagłówków (<th> oraz </th>) . Poniżej przykład obrazujący prostą tabelę o nagłówkach „id" i „typ"[7]:

<tr><th>id</th><th>typ</th></tr>
<tr> <td>1</td> <td>przykładowy typ 1</td> </tr>
<tr> <td>2</td> <td>przykładowy typ 2</td> </tr>

co odpowiada poniższej tabeli:

id	typ
1	przykładowy typ 1
2	przykładowy typ 2

Załącznik A: Obliczanie rozmiaru kodu w zależności od jego wersji

Rozmiar modułu kodu w wygenerowanym obrazku 600 DPI (24 DPmm) zbudowanego z 6 x 6 punktów zajmuje 0,25 x 0, 25 [mm].

Załóżmy, że podczas generacji kodu danymi ze skierowania, kod wynikowy jest w wersji 17. Posiłkując się tabelą ze strony http://www.denso-wave.com/qrcode/vertable2-e.html, wiemy, że kod taki zbudowany jest z 85 x 85 modułów. Rozmiar kodu wynikowego wobec tego wyniesie w szerokości i wysokości: 85 * 0,25 = 21,25 [mm].

Można również odwrotnie podejść do sprawy i obliczyć jaki rozmiar będzie miał kod dla określonej objętości danych. Wiedząc, ze stopień korekcji błędów wynosi M, oraz znając ilość znaków to posiłkując się tabelą z powyższego linku można uzyskać informację o wersji kodu. Znając wersję i rozmiar modułu można wyliczyć wynikowy rozmiar kodu.

Załącznik B: Wygenerowanie kodu 2D z przykładowego zestawu danych

[6] Znaki nowej linii na końcach wierszy są wprowadzone dla czytelności i nie powinny się znaleźć w polu danych.
[7] Znaki nowej linii na końcach wierszy są wprowadzone dla czytelności i nie powinny się znaleźć w polu danych.

Przykład 1:

W tabeli poniżej znajduje się przykładowy zestaw danych testowych, dla którego odbędzie się generowanie kodu:

Nr pola danych	Dana
1	Jan Kowalski
2	12122012
3	ąśżźćńłóĄŚŻŹĆŃŁÓ
4	%^&*() ?`}|
5	Koniec

Typ danych oznaczony jako „11” w wersji „09”.

Poniższy ciąg znaków przedstawia ramkę z danymi przed mapowaniem znaków na liczby. Separator pól danych oznaczono przez „**SEP**”, znak „Shift” oznaczono przez „**SHIFT**”, znak „Caps Lock On” oznaczono przez „**CLON**”, a „Caps Lock Off” przez „**CLOFF**”. Sekwencje początkowa o końcowa oznaczone kolorem niebieskim.

@^`1109**SHIFT**Jan **SHIFT**Kowalski**SEP**12122012**SEP**ąśżźćńłó**CLON**ĄŚŻŹĆŃŁÓ**SEP**%^&*() ?`}|**SEP**K**CLOFF**oniec@@@

Po mapowaniu znaków na liczby otrzymano poniższy ciąg liczbowy:

64949649494857977465783297757987657683757310495049505048495010111718191215141698111718191215141610379438424041326396222107599797873696764646 4

Poniżej wygenerowany kod:

Powyższy kod posiada następujące parametry:
Kodowanie: numeryczne
Korekcja błędów: M
Wersja kodu: 4
Liczba modułów: 33 x 33

Po przeczytaniu kodu i przemalowaniu liczb na znaki otrzymamy zestaw danych taki sam jak w tabeli przykładowych danych.

Przykład 2:

Poniższa tabela przedstawia bardziej skomplikowany zestaw przykładowych danych. Oprócz umieszczenia w nim listy, założono że pierwsze 3 pola znajdą się w pierwszym kodzie, natomiast pozostałe znajdą się w drugim.

Nr pola danych	Dana
1	Jan Kowalski

2	<di> 12122012 - Pracownik ochrony </di>
3	ąśżźćńłóĄŚŻŹĆŃŁÓ
4	%^&*() ?`}|
5	Ostatnie pole i koniec 1234567890

Typ danych oznaczony podobnie jak w przykładzie 1. Identyfikator kodu (liczba losowa): 712345.

Poniższy ciąg znaków przedstawia ramkę z danymi przed mapowaniem znaków na liczby. Separator pól danych oznaczono przez „**SEP**”, znak „Shift” oznaczono przez „**SHIFT**”, znak „Caps Lock On” oznaczono przez „**CLON**”, a „Caps Lock Off” przez „**CLOFF**”. Sekwencje początkowa o końcowa oznaczone kolorem niebieskim.

Kod 1:
@^`E217123451109**SHIFT**Jan **SHIFT**Kowalski**SEP**<di><li>12122012</li><li>**SHIFT**Pracownik ochrony</li></di>**SEP**ąśżźćńłó**CLON**ĄŚŻŹĆŃŁÓ`^@

Kod 2:
`^@E22712345%^&*() ?`}|**SEPSHIFT**Ostatnie pole i koniec 1234567890@@@

Po mapowaniu znaków na liczby otrzymano poniższe ciągi liczbowe:

Kod 1:

64949669504971234549494857974657832977579876576837573**10**6068736260767362495049505048495060477673626076736297808265677987787375327967728279788960477673626047687362**10**1117181912151416981117181912151416969464

Powyższy kod posiada następujące parametry:
Kodowanie: numeryczne
Korekcja błędów: M
Wersja kodu: 6
Liczba modułów: 41 x 41

Kod 2:

9694646950507123453794384240413263962221**10**97798384658478736932807976693273327579787369673249505152535455565748646464

Powyższy kod posiada następujące parametry:
Kodowanie: numeryczne
Korekcja błędów: M
Wersja kodu: 4
Liczba modułów: 33 x 33