PAY by square specifications

specifications for PAY by square standard

by square®

www.bysquare.com
info@bysquare.com

Disclaimer

This document represents the PAY by square specifications created by ADELANTE, s.r.o. for the benefit of Slovenská banková asociácia (SBA) to be the standard for encoding and decoding of payment information into and from QR codes for the members of the SBA. The contents herein are intended for free public circulation, however, they are not to be changed in any manner, in whole or in part without the prior express written permission of the SBA. Unauthorised interference into the content of this document and its subsequent reproduction, printing, publishing, posting, displaying, incorporation, storing in or scanning into a retrieval system or database, transmission, broadcasting, bartering or selling is strictly prohibited and is an infringement of copyright laws.

Foreword

ADELANTE, s.r.o. is a Slovak Republic based corporation developing, publishing, promoting and aiding implementation of the PAY by square standard for the benefit of the SBA. This document represents the PAY by square specifications. It is intended to serve as basis for evaluation and implementation of the PAY by square standard by developers and professional community.

Document overview

Document version and history

Table 1 – document version and history:

Version	Release Date	Note
0	2013-02-22	created this document from original by square specifications
1	2015-06-24	added fields for beneficiary name and address

by square - PAY specifications - 1.1.0

Table 2 - List of files which are part of PAY by square specifications 1.1.0:

File name	Note
by square - PAY specifications - 1.1.0.pdf	***this document***
bysquare-schema-pay-1.1.0.zip	XSD document of the by square data model
bysquare-schema-doc-pay-1.1.0.zip	PAY by square schema HTML documentation
by square - PAY logo manual - 1.0.0.pdf	PAY by square logo manual

Table 3 - PAY by square schema version specified in this document:

by square schema	Version
by square schema - pay	1.1.x

Table 4 - by square type versions specified in this document:

by square type	Version
PAY by square	0

Using this document

Artifacts of the by square standard

To define by square standard and aid its implementation, ADELANTE, s.r.o. specifies and releases number of artifacts. Here is the list of these artifacts:

- by square specifications
- by square schema
- by square software libraries
- by square implementation manual
- by square test utilities
- by square encoding/decoding applications

by square specifications comprises of several documents (including this document) and provides all information on the by square standard required for successful implementation of the by square standard.

by square schema defines the data model of the by square standard. It is represented as an XSD document, which specifies data structure of all by square types.

by square software libraries are software libraries developed by ADELANTE, s.r.o., to aid implementation of the by square standard. by square software libraries significantly decrease costs of implementing by square and help achieve consistent by square implementations across businesses and banks.

by square implementation manual provides necessary documentation to by square software libraries.

by square test utilities provide means for efficient testing of your implementation. by square test utilities significantly decrease costs of implementing by square and help achieve consistent by square implementations across businesses and banks.

by square encoding/decoding applications help you try, test and evaluate capabilities and functions of the by square standard.

Versioning

by square type versioning is a sequence based versioning using one sequence, denoting version of the by square type. This sequence is increased when data model of the by square type, as defined by the by square schema, is modified. Whenever by square type version is increased, new versions of by square standard schema, by square software libraries and documentation are released.

by square schema versioning is a sequence based versioning using three separate sequences with individual meaning:

major.minor.bug(revision)

Major sequence is increased when new by square type is added, Minor sequence is increased when data model of any of the existing by square types is modified and Bug (revision) sequence is increased when bug or error is fixed in current release of the schema.

by square software libraries versioning is a sequence based versioning using three separate sequences with individual meaning:

major.minor.bug(revision)-optional(feature)

First two sequences - major and minor refer to the versioning of the by square schema. This means that the by square schema and related software libraries have the same major.minor version number. Bug (revision) sequence is increased when bug is fixed or revision is done to the current release of the software libraries. Optional(feature) is a string indicating a new feature that does not necessarily mean a new version. If not using this feature, upgrade is not necessary.

by square specifications versioning is a sequence based versioning using three separate sequences with individual meaning:

major.minor.bug(revision)

First two sequences - major and minor refer to the versioning of the by square schema. This means that the by square schema and corresponding specifications have the same major.minor version number. Bug (revision) sequence is increased when error is corrected or revision is done to the current specifications.

Example: Current by square type PAY by square is in version 0.

Data structure of these by square types is defined in the current version 1.1.0 of the PAY by square schema. PAY by square specifications 1.1.x and by square software libraries 1.1.x are all compatible with PAY by square schema 1.1.0. This being said we always recommend use of the latest specifications and software libraries.

Terminology

ASCII - character encoding scheme based on english alphabet

Base32hex - binary-to-ASCII encoding schema

BSQR - by square QR code

by square code logo - identifies QR code 2005 symbol encoded according to by square Standard

by square type logo - identifies compatibility with by square Standard

by square type - subset of the by square standard, intended for specific functional domain.

client data - any data serving as input for encoding

client attribute - a particular entity in the client data

document type - well defined and structured collection of client data required for a specific act or task within given functional domain. PAY by square has exactly one document type and that is PAY.

QR code 2005 - international standard defined in [1]

QR code 2005 symbol - matrix consisting square light and dark modules arranged in an overall square pattern

CRC32 checksum - error detection mechanism for data

LZMA - an algorithm for data compression

Introduction

PAY by square standard is a standard optimized for electronic encoding of payment orders into a QR code and decoding payment orders from a QR code. Nevertheless by square format can be used for other purposes such as NFC.

Throughout this document we are referring to QR codes generated according to by square standard as by square QR codes or BSQR codes for short. We will refer to payment data as client data.

Purpose and scope

By square is a standard defining electronic data format and process of encoding and decoding client data into and back from a QR Code 2005 symbol. The purpose of this specifications document is to explain the underlying electronic format and process. This document specifies a client data attribute list, attribute metadata, W3C XML schema for data validation, CRC32 algorithm for checksum generation, LZMA algorithm for data compression, Base32hex algorithm for binary-to-ASCII data conversion and QR Code 2005 input parameter settings. This document also discusses error detection and QR Code 2005 symbol size considerations. The overall aim is to provide a detailed description of each step of the encoding and decoding process.

1. Requirements and conformance

The specification makes use of **XML** and **XSD** schema as recommended by W3C consortium [2,4]. A Lempel–Ziv–Markov chain compression algorithm - **LZMA** is used for compression of the client data. **CRC32** a cyclic redundancy check, an error detecting algorithm and code is used to ensure data integrity of client data. A **Base32** algorithm [3] is used for alphanumeric encoding. The graphical result of the by square system is produced by **QR Code 2005** international standard [1]. The by square documentation is compliant with all mentioned standards.

2. Overall description

The next section of the document (see section 3) describes the encoding and decoding steps of the by square standard (fig. 1). The client data for encoding is presented as an XML document. The structure of the XML must conform an XML schema which is part of the overall specification. The data is read, processed (as described in 3) and ordered into a data sequence. Afterwards a CRC32 checksum is computed, appended at the head of the data sequence and both are compressed by the LZMA algorithm. A header with metadata is appended to the compressed data. The resulting data sequence is translated to an ASCII representation using a specific Base32hex scheme (described in 3.13). The ASCII representation serves as an input for the QR Code 2005 encoder which produces the symbol. The decoding process described in section 3.16 consists of the same steps as the encoding process. The individual decoding steps represent an inverse operation to their encoding counterpart.

3. Description

This section describes the PAY by square Standard and the process of encoding and decoding client data to and from a QR Code 2005 symbol as described in section 2 Overall Description and on fig. 1.

3.1. PAY by square type

PAY by square is part of the by square standard developed by ADELANTE, s.r.o. for the benefit of Slovenská banková asociácia (SBA). The by square standard comprises of several by square types. Each by square type is designated for a specific functional domain. This document covers **only** PAY by square type.

Table 5 - Full overview of by PAY by square type and documents

by square type	by square type bit value	version	version bit value	document type	document type bit value
PAY by square	0000	0	0000	pay	0000

3.2. PAY by square logo

by square code logo purpose is to clearly identify by square code belonging under given by square type.

It is mandatory, that by square code logo always accompanies BSQR code either in color or black and white graphical representation. All permitted variations of the by square code logo are in the by square - logo manual which is a part of these specifications.

Table 6 - overview of the basic by square code logos and type logos

by square Type	by square code logo	by square code logo (optional electronic use)
PAY by square		

3.3. PAY by square versions

Purpose of the version is to differentiate latest definition of the PAY by square type from older definitions. This is important to ensure correct interpretation of the by square codes and to ensure backward compatibility.

PAY by square type version is clearly indicated:

- in this specifications document in section document overview,
- in the by square schema XSD document, for each by square type (root element),
- in the header of each by square code (as described in the section by square header).

Highest supported PAY by square type version should be also clearly indicated in all applications conforming with by square standard.

3.4. Document types

by square type can further contain several document types. Document types are well defined and structured collections of client data required for a specific act or task within given functional domain. PAY by square type contains only one document type.

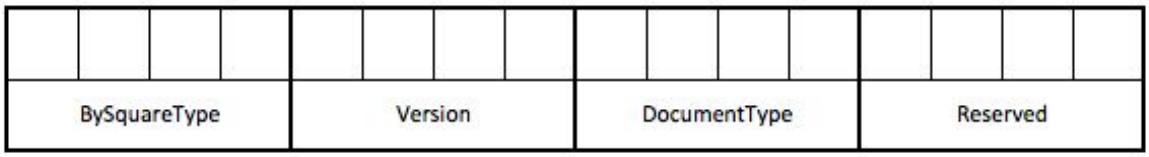
3.5. by square header

by square type, version and document type constitute the header of each message within the by square standard. Message header ensures that recipient of the BSQR code has instant information on what type of information it is receiving and whether he is capable of decoding the information.

Table 7 – by square Header

Attribute	Number of bits	Possible Values	Note
BySquareType	4	0-15	by square type
Version	4	0-15	version of the by square type
DocumentType	4	0-15	document type within given by square type
Reserved	4	0-15	bits reserved for future needs

Figure 2 – by square header overview



3.6. Supplying an XML document with client data

For purpose of this document, input client data is represented as an XML 1.0 document according to the W3C specification standard [2]. The XML standard allows structuring of data at the necessary level in order to validate and verify it for completeness. The client XML document must be in conformance with a XML schema which is part of the specification. The by square schema describes a complete list of possible attributes, fields, values and classifiers for each document type under each by square type.

For purpose of particular implementation, input client data can have any representation, but need to meet criteria set forth in the XML schema, such as data format, length, order, etc..

3.7. Preprocessing of client data

Dates, numbers, codes

Date fields are provided according to XML schema. After reading field values from client XML document all date field values (see section 4 for more on data model) are converted from (ISO 8601) YYYY-MM-DD to YYYYMMDD format. The result of the conversion is considered as a new value for the given date field.

Numbers, currency codes, country codes, BIC and IBAN need to be provided in specified format according to table 8.

Table 8 - Data formatting rules

Data Type	Format	Description	Example
date	YYYYMMDD	ISO 8601	'2012-03-14' for 14th of March 2012
decimal number	#.########	dot is used as decimal mark	3.14159265
currency code	XXX	3 letter representation ISO 4217	EUR
country code	XXX	3 letter representation ISO 3166 (alpha-3 code)	SVK
BIC	ISO 9362	8 or 11 characters long bank identifier code	TATRSKBX
IBAN	ISO 13616	up to 34 characters long international bank account number	SK5111000000002922782343

Multiple options

If a field has multiple options assigned, the classifiers of these options are summed and the resulting integer is considered as the new value of the field. For example we have a month field with 1 (January), 64 (July) and 512 (October):

Input: 1, 64, 512

Output: 577

List and detailed description of such attributes is given in Appendix A.

3.8. Generating field sequence

After data is read from the XML document and preprocessed as described in the previous section the attribute values are ordered into a sequence. Following rules apply:

- The order in which the field values are put into the sequence is defined by the order attribute in the XSD schema. If any optional field value defined in the schema is missing the value is treated as if it had 0 length. Between each value pair a ASCII horizontal tab character is inserted. In order to ensure the sequence format is correct, any horizontal tab characters found in field values are replaced with a space character. Values of length 0 are also considered and it may result in two or more consecutive tab characters. The horizontal tab character represents a separator between any two field values and the correct field ordering can be reconstructed after decoding takes place.

- If a field has schema attribute maxOccurs="unbounded", this means that such field can repeat in the sequence. Before adding first instance of such repeating field into the sequence, an integer number representing the number of occurrences is inserted into sequence and delimited with horizontal tab character. This is important for proper decoding of the sequence, as this integer number informs decoder on how many occurrences of given attribute it should expect. *Example is: BankAccount element*
- If a field is a complex type and has schema attribute minOccurs="0", this means that the whole complex element might not appear in the sequence. Before adding such attribute into the sequence, an integer number of 0 or 1, representing number of occurrences is inserted and delimited with horizontal tab character. 0 represents no occurrences of given complex element, element children are ignored in such case and the whole process continues with the next attribute in the schema. *Example is: DirectDebitExt complex element*

From this point further it is convenient to perceive the client data not as a character sequence but as a UTF-8 bit sequence and we will refer to is as such further on.

3.9. Sequence length

3.9.1. Data sequence length for encoding into QR codes

Due to the data size limit of a QR code 2005 symbol, the string sequence consisting of preprocessed client data has to be limited. Setting the sequence limit is subject to various factors. Too large limit will result in poorly readable QR code symbols when printed to a small physical size. Too small limit will result in too few data contained in the QR code symbol. After careful considerations of the physical size and the average size of data, the sequence limit was set to **550 UTF-8 characters** including the delimiter (horizontal tab character). This length limit is sufficient enough to contain data for up to 5 payment orders with multiple bank accounts.

Any combination of payment data, whether it is multiple payments, various payment types or multiple bank accounts can be combined into single code, as long as the resulting UTF-8 bit sequence is shorter than 550 characters. If sequence should exceed the limit of 550 characters, data should be split into multiple codes or some data (such as alternative bank accounts) should be omitted.

3.9.2. Data sequence length for other purposes

For other use than encoding into QR codes, the sequence length limit can be disabled and single by square code can contain theoretically unlimited number of payments and bank accounts. However the data is limited to 65535 compressed bytes due to the implementation of compression algorithm discussed in 3.11.

3.10. Appending CRC32 checksum

Before compression takes place a CRC32 checksum is computed from the bit sequence and the checksum is appended to the beginning of the bit sequence, as the first 32 bits. The following polynomial is used in the CRC32 algorithm:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

When decoding by square codes, this stored checksum is compared with the calculated checksum of the decompressed data to ensure data integrity.

3.11. LZMA Compression

The compression algorithm used is LZMA. The header of compressed data is 2 bytes long and contains only one 16-bit unsigned integer (word, little-endian), which is the size of the decompressed data. Therefore the maximum size of compressed data is limited to 65535. The parameters for compression are:

- LC = 3
- LP = 0
- PB = 2
- DICTIONARY_SIZE = 128 kilobytes (2 ^ 17)

After decompression, the first 4 bytes of the decompressed data is CRC32 checksum (DWORD, little-endian) of the decompressed data.

3.12. Appending header information

In the next step a data header is appended to the bit sequence. The header consists of four, four bit unsigned integers representing the header values – by square type, version, document type and reserved bits. Table 7 describes header structure and Table 5 describes header values.

3.13. Alphanumeric conversion using Base32hex

In order for the Base32hex to properly encode the bit sequence (result of compression in 3.11) the bit count must be divisible by 5. For this purpose Base32hex adds padding bits at the end of the sequence so that resulting sequence is divisible by 5. (When decoding such sequence, padding bits will be truncated to the closest length divisible by 8.)

The bit sequence will be transformed into ASCII character sequence which complies with the character set of the QR Code 2005 alphanumeric encoding mode. The transformation is carried out by the Base32hex which complies to basic Base32 transformation algorithm [3] with a specific (hexadecimal) mapping scheme defined in table 9 and specific padding bits treatment described above. The bit sequence is split into 5 bit chunks which are mapped onto the characters in table 9.

Table 9 – Encoding table

Character	4 (MSB)	3	2	1	0 (LSB)
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
A	0	1	0	1	0
B	0	1	0	1	1
C	0	1	1	0	0
D	0	1	1	0	1
E	0	1	1	1	0
F	0	1	1	1	1
G	1	0	0	0	0
H	1	0	0	0	1
I	1	0	0	1	0
J	1	0	0	1	1
K	1	0	1	0	0
L	1	0	1	0	1
M	1	0	1	1	0
N	1	0	1	1	1
O	1	1	0	0	0
P	1	1	0	0	1
Q	1	1	0	1	0
R	1	1	0	1	1
S	1	1	1	0	0
T	1	1	1	0	1
U	1	1	1	1	0
V	1	1	1	1	1

MSB - Most significant bit, LSB - Least significant bit

The mapping in table 9 takes advantage of the alphanumeric encoding mode specified by [1]. Note that the resulting character sequence takes roughly 10% more space than the original bit sequence.

Please note that the resulting Base32hex character sequence can be directly used for NFC, email attachment, etc.. or can be encoded into QR Code as described in the next section. To store the Base32hex sequence into a file, use the extension "**.bsqr**" at the end of the filename for proper identification of the file content.

3.14. Generating by square Code

The character sequence generated by the Base32hex encoding represents the input for the QR Code 2005 encoder. by square standard is using QR code 2005 to graphically represent encoded data in form of a 2D code. QR Code 2005 is a matrix symbology specified by ISO/IEC [1]. The symbols consist of an array of nominally square light and dark modules arranged in an overall square pattern. QR code specifications define 40 versions of QR code symbol and provide rules for evaluating physical size of the QR code symbol, required for reliable and fast processing by various devices. by square standard was designed so, that the individual by square types never exceed specific QR code symbol version and so a fixed physical size for a QR code symbol can be defined. This means, that no matter which QR symbol version is encoding by square data, the symbol is printed in fixed physical size. Fixed physical size greatly simplifies layouting of invoices and payment orders and guarantees smooth processing of by square codes by all common devices.

Full overview of maximum symbol versions, physical sizes of QR code symbol and physical sizes of by square codes is given in table 10.

Table 10 - QR code symbol physical size

by square Type	Max sequence length (characters)	Recommended	Minimum
PAY by square	550	36mm x 36mm	30mm x 30mm

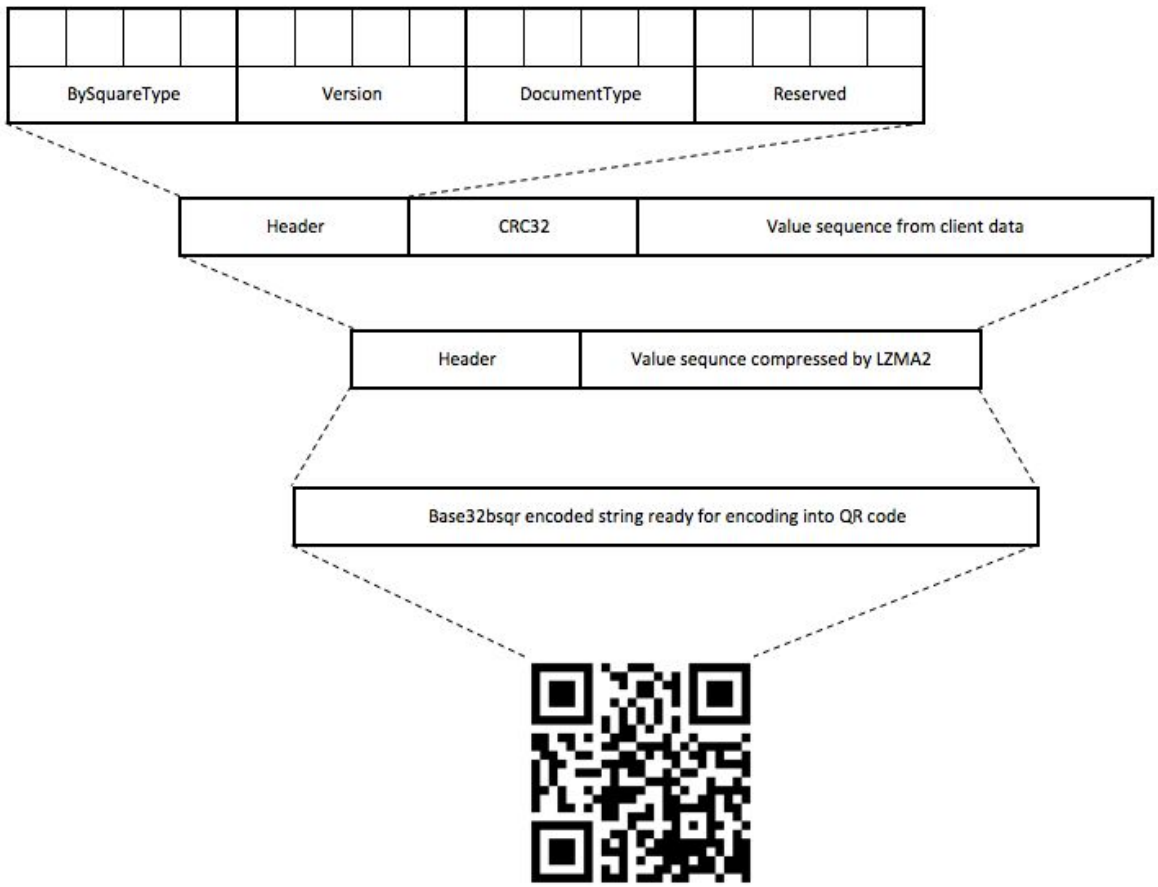
QR code encoding parameters are listed in table 11. To achieve minimum possible version of the QR code, it is instrumental to use Alphanumeric mode for encoding by square.

Table 11 – QR code 2005 encoding parameters used by by square

Parameter	Value	Note
code version	xL	x is number (from 1 to 17)
mode indicator	Alphanumeric	0–9, A–Z (uppercase only), space, $, %, *, +, -, ., /, :

3.15. Summary – encoding process overview

Figure 3 – by square encoding process overview



3.16. Decoding client data from QR Code 2005 symbol

The decoding process is straightforward and is comprised of the inverse operations described in the encoding section.

The QR Code 2005 symbol is decoded according to the standard [1]. The output is a character sequence which must be further decoded using the Base32hex scheme defined in table 9. Note that header information is stored in the first four ASCII characters of the sequence which if necessary can be manipulated independently. The result of the Base32hex decoding is a bit sequence with the structure on fig. 2 together with the structure of the header. First 4 bits identify the type of client data which is being decoded, possible values are in table 5. Next 4 bits hold the version number, which must be equal to (or lower than) the version number of the decoder, otherwise the decoding process terminates with an unknown document error. Version bits are followed by 4 document type bits which further specify which subtype of by square Type is being decoded (see table 5 for a complete list). Note that by square type and document type uniquely identify the attribute list encoded in the QR Code 2005 symbol. The header ends with 4 bits that are reserved.

After interpreting the header information and truncating padding bits, the data part is decompressed by the LZMA algorithm. The output of decompression is the CRC32 checksum and the sequence of field values. To verify client data integrity a CRC32 checksum is computed from sequence of values as described in 3.10. We compare the computed and saved CRC32 (see fig. 2) checksums. In case of a mismatch the decoding process is terminated, otherwise decoding continues. Field values in the sequence are separated by the ASCII horizontal tab character. If a field value has not been supplied in the input XML document the field value has 0 length. This results in two consecutive tab characters or a missing last attribute value in the sequence. All 0 length values are ignored. Order of values in the sequence matches to the order of elements specified in the XSD schema. Through the correspondence of the ordering values can be mapped to their fields. After establishing the field to value mapping the date fields and multiple option fields are restored (see section 3.7). All values of date fields are transformed from YYYYMMDD back to YYYY-MM-DD format and the value in the latter format is considered as the new value of a given date field.

All field values with multiple options are represented as a sum of classifier integers as described in the XSD schema. Restoring the original values is achieved by decomposing the sum into the individual classifier integers. Decomposition of the sum is done by the following algorithm:

SET $classifiers = \{\}$
FOR($i = 1; i \leq \#options; i++$)
BEGINFOR
  $next = 2^{\#options - i}$
  IF($sum - next > 0$)
  BEGIN
    $sum = sum - next$
    $classifiers = classifiers \cup \{next\}$
  END
ENDFOR

Where #options is the number of all possible values of the given attribute and sum is the initial value of the attribute.

The final step is to create a XML document with the desired fields. The resulting XML document must comply with the XML schema which is a part of the overall documentation.

4. PAY by square datamodel

PAY by square data model is specified in the schema document. Few points should be carefully observed:

Longer bulk payment orders which exceed the capacity of one PAY by square Code are encoded into multiple PAY by square codes, where each PAY by square code is treated as individual bulk payment.

Appendix A - classifiers.

Appendix A contains lists of all classifiers used within by square standard.

Table 9 – Periodicity classifier

Value	Periodicity
d	Daily
w	Weekly
b	Biweekly
m	Monthly
B	Bimonthly
q	Quarterly
s	Semiannually
a	Annually

Table 10 – Month classifier

Value	Month	Computation
1	January	2^0
2	February	2^1
4	March	2^2
8	April	2^3
16	May	2^4
32	June	2^5
64	July	2^6
128	August	2^7
256	September	2^8
512	October	2^9
1024	November	2^10
2048	December	2^11

Table 11 – Payment options classifier

Value	Payment option	Computation
1	paymentorder	2^0
2	standingorder	2^1
4	directdebit	2^2

Table 12 - Direct debit type

Value	Direct debit type
0	one-off
1	recurrent

Table 13 - Direct debit scheme

Value	Direct debit scheme
0	other
1	SEPA

Appendix B - encoding recommendations.

Appendix B contains recommendations for implementing encoding of the by square codes.

Implementation philosophy

Philosophy behind the by square standard is simplicity. The process of scanning and utilizing by square codes should require the least possible number of steps. To meet this goal, it is instrumental that by square codes are carefully and properly encoded.

List of encoding recommendations

1. general recommendations
2. recommendations for encoding PAY by square for invoice

1. General recommendations

1. on using by square software libraries
2. size of the QR code module
3. PAY by square logo

1.1. On using by square software libraries

By square recommends use of by square software libraries for encoding and decoding by square codes. XML document is the recommended option for providing input data when encoding by square codes and for obtaining outputs when decoding by square codes. Across Appendix B we assume use of these software libraries and XML files. As a companion to this document we are providing sample xml files which illustrate the standard use of by square.

1.2. Size of the QR code module

Recommended and minimum size of QR code with by square logo is defined in by square logo manual. This size of the QR code guarantees smooth readability of the by square code with all standard devices including smart phones.

1.3. PAY by square logo

PAY by square QR code needs to be appended with PAY by square logo. Purpose of this is to provide user with instant recognition of the by square QR codes and to guarantee consistent and smooth user experience.

2. Recommendations for generating PAY by square for invoice

1. encoding payment options
2. encoding bank accounts
3. encoding payment note
4. other payment order considerations
5. other standing order considerations
6. other direct debit considerations

2.1. Encoding payment options

Three types of payment can be encoded into PAY by square code: (simple) payment order, standing payment order and direct debit. To clearly define what has been encoded into PAY by square code **PaymentOptions** field need to be filled in. Based on the **PaymentOptions** field, applications decoding PAY by square code can present user with the options available. Multiple payment options can be combined together, for example to create a payment order with sanding order extension, simply fill in “paymentorder standingorder”. The values must be separated by a space character. To find out which payment options are supported by the banks see the following web site: http://www.sbaonline.sk/sk/.

PaymentOptions field values are:

- “paymentorder”
- “standingorder”
- “directdebit”

2.2. Encoding bank accounts

Bank accounts are encoded in the IBAN and BIC format. You can encode several various bank accounts into PAY by square code to give user various options where he can route the payment. The default bank account should be encoded as the first in the sequence. Bank applications decoding PAY by square code will either chose preferred (own) bank account or will use default (first) option.

The exact number of bank accounts that can be encoded into single PAY by square code is limited by the code capacity and depends on other payment options. For a simple payment order there can be encoded up to 6 bank accounts.

2.3. Encoding payment note

We recommend that the **PaymentNote** field is prefilled for the user by the issuer of the invoice and payment. This helps user to keep clear track of their payments and expenses, without the need to type their custom payment note. To provide consistent and structured information in the **PaymentNote** field, we recommend payment note structure as defined in table 14.

Table 14 – PaymentNote structure

Issuer	Delimiter 1*	Subject	Delimiter 2**	Client	Delimiter 3**	Term***
name of the issuer of the invoice or payment	-	subject of the invoice or payment	-	name of the client	-	term of date of the service or product delivery
recommended		optional		optional		optional

**delimiter 1 is colon and space „: “*

***delimiter 2 and delimiter 3 is space dash space „ – „*

****recommended formats for the term are: YYYY, YYYY/MM, YYYY/MM/DD, YYYY/QQ*

Examples of the payment note:

Telefonica: mobile internet – 2012/03

Law Firm: legal services – 2012/1Q

Amslico: life insurance – Mr. Podhajsky – 2012

2.4. Other payment order considerations

When encoding simple payment order please consider following fields:

- **PaymentOptions** – needs to be filled in with “paymentorder” option
- **Amount** – encoded with amount payable. This field is not required and can be left blank in cases payment amount is not known - such as donations
- **CurrencyCode** – 3 letter, payment currency code according to ISO 4217
- **PaymentDueDate** – optional field
- Payment reference can be encoded in one of the following two ways: 1. by using three optional fields **VariableSymbol**, **SpecificSymbol** and **ConstantSymbol** 2. by encoding **OriginatorsReferenceInformation** which is defined under SEPA.
- **PaymentNote** – optional field. In previous section we provide further recommendations for encoding payment note
- **BankAccounts** - In section „encoding BankAccounts“ we provide further recommendations for encoding bank accounts

2.5. Other standing order considerations

Standing order is encoded as an extension to payment order. This means, that only the additional information required to define standing order is provided. When encoding standing order please consider following fields:

- **PaymentOptions** – needs to be filled in with „standingorder“ option
- **Day** – this is the payment day. It‘s meaning depends on the periodicity, meaning either day of the month (number between 1 and 31) or day of the week (1=Monday, 2=Tuesday, …, 7=Sunday).
- **Month** – selection of one or more months on which payment occurs. This is enabled only if periodicity is set to one of the following value: “Weekly, Biweekly, Monthly, Bimonthly”. Otherwise it must not be specified.
- **Periodicity** – periodicity of the payment. All valid options are „Daily“, „Weekly“, „Biweekly“, „Monthly“, „Bimonthly“, „Quarterly“, „Annually“, „Semiannually“. To find out which periodicity types are supported by the banks see the following web site: http://www.sbaonline.sk/sk/
- **LastDate** – defines the day of the last payment of the standing order. After this date, standing order is cancelled.

2.6. Other direct debit considerations

Direct debit is encoded as an extension to payment order. This means, that only the additional information required to define direct debit is provided. When encoding direct debit please consider following fields:

- **PaymentOptions** – needs to be filled in with „directdebit“ option
- **DirectDebitScheme** – this field can have “SEPA” value, if direct debit is using SEPA direct debit scheme or “other” when an ordinary direct debit is defined
- **DirectDebitType** – can be „one-off“ for one time debit or „recurrent“ for repeated debit until cancelled.
- Direct debit reference can be encoded in one of the following three ways: 1. by using two optional fields **VariableSymbol** and **SpecificSymbol**. 2. by encoding only **OriginatorsReferenceInformation**. 3. by using two required fields **MandateID**, **CreditorID** and one optional field **ContractID** identifying direct debit under the SEPA direct debit scheme.
- **MaxAmount** – optional field. As most users prefer to set up some maximum amount for the direct debit, this can be pre-filled for them
- **ValidTillDate** – defines the day after which direct debit is cancelled.

Appendix C - decoding recommendations.

Appendix C contains recommendations for implementing decoding of the by square codes.

Implementation philosophy

Philosophy behind by square standard is simplicity. The process of scanning and utilizing the by square codes should require the least possible number of steps. This philosophy should be preserved when implementing by square standard into applications. The process of decoding by square codes should ideally require just two user actions:

- User opts for scanning by square code (ex. PAY by square code)
- User validates the prepared actions (ex. user authorizes payment for execution)

List of decoding recommendations

1. recommendations for mobile banking applications
2. recommendations for decoding PAY by square

1. Recommendation for mobile banking (smart banking) applications

List of main functionality to be considered in mobile banking applications:

1. scanning PAY by square code by camera
2. extracting PAY by square from a pdf or picture file
3. generating PAY by square for a peer to peer application
4. generating PAY by square and saving or sending the picture or PDF file by email

On mobile devices, the action of extracting PAY by square code from a PDF or picture file would be ideally triggered by sending PDF or picture file into the payment app, which would automatically search for the presence of the by square code. Such implementation would ensure, that user can pay any invoice received as an attachment to an email with only two actions (send PDF to banking application and authorize payment). Further we refer to both actions 1 and 2 as scanning PAY by square code.

List of minor functionality:

1. provide user with option to modify and append payment information
2. providing user with choice when multiple payment options are available (ex. choice to either execute one time payment or to set up standing order)
3. executing bulk payment orders (either one by one or as a true bulk payment order with one authorization)
4. storing InvoiceID when payment was executed and notifying user next time when he attempts to process the same payment

5. give user option to execute payment at the very moment or postpone payment execution until PaymentDueDate. For better user experience, this can be made general setup option not a payment by payment choice.
6. notify user if payment is past it's PaymentDueDate and suggest execution of payment at the very moment

2. Recommendations for decoding PAY by square

2.1. Recommendations for filling in payment order from PAY by square Code

Filling in payment order from a PAY by square Code is trivial. All fields of payment are used, excluding StandingOrderExt and DirectDebitExt. Three points should be carefully implemented:

1. treatment of **PaymentDueDate**
2. selection of **BankAccount** if multiple bank accounts are available. First bank account on the list is considered to be the default bank account. We recommend that the application chooses either the bank account at the own bank or the default bank account.
3. treatment of **PaymentNote**, which might be provided in the PAY by square Code for user convenience. User should have the option to keep proposed PaymentNote or to modify it

2.2. Recommendations for filling in standing order from PAY by square Code

Standing order is treated as an extension of payment order. To fill in standing order form from a PAY by square code requires taking all fields of the payment order and of standing order extension. These points should be carefully implemented:

1. **PaymentDueDate** = first payment date of the standing order. If this is not supported by your standing order system – split the execution into standard payment order for the first payment and the standing order for the remaining payments. Carefully treat cases when PaymentDueDate is in the past.
2. selection of **BankAccount** if multiple bank accounts are available. First bank account on the list is considered to be the default bank account. We recommend that the application chooses either the bank account at the own bank or the default bank account.
3. treatment of **PaymentNote**, which might be provided in the PAY by square Code for user convenience. User should have option to keep proposed PaymentNote or to modify it
4. periodicity. Implementing periodicity is relatively complex. The most important parameter is **Periodicity**. It defines when do consequent payments occur, starting from first PaymentDueDate. Such as daily, weekly, monthly, quarterly, annually. If Periodicity is set to Weekly, Biweekly, Monthly or Bimonthly then it is possible to use field **Month** to choose only selected months of the year for payment execution. If Periodicity is set to other than daily, weekly and biweekly, field **Day** can be filled in to specify on which day of the month the payment should occur. If Periodicity is set to

weekly or biweekly, field Day can be filled in to specify on which day of the week the payment should occur.

5. **LastDate,** which specifies the day of the last payment of the standing order. If this is not supported by your standing order system, application should notify user that he needs to cancel standing order on that day.

2.3. Recommendations for filling in direct debit order from PAY by square Code

Direct debit is treated as an extension of payment order. To fill in standing direct debit form from a PAY by square Code requires taking some fields of the payment order and of direct debit extension. These points should be carefully implemented:

1. **PaymentDueDate** = start date of the direct debit validity. If this is not supported by your standing order system set up the direct debit with validity as of current date
2. Fields **MaxAmount** and **ValidTillDate** are optional fields which are pre-filled for customers convenience when setting up direct debit with limit on maximum amount which can be debited and with direct debit termination on a scheduled date if such is foreseen.
3. **DirectDebitScheme**. If DirectDebitScheme value is 1, which is „SEPA“ than encoded direct debit follows SEPA direct debit scheme which means that fields MandateID, CreditorID and optional ContractID are used. If direct debit scheme is 0, which is „OTHER“ this means no specific direct debit scheme and following rules do apply:
    a. Creditor is identified via bank accounts
    b. Contract between debtor and creditor is identified using one of the following two ways: 1. by two optional fields **SpecificSymbol** and **VariableSymbol.** 2. by one optional field **OriginatorsReferenceInformation**. If SpecificSymbol and VariableSymbol fields or OriginatorsReferenceInformation field is filled in DirectDebitExt then these fields do apply for the direct debit.

Appendix D - data model overview

Table 15 - PAY by square sequence data model

Order	Field name	Type	Priority	Maximum length
1	InvoiceID	string	2	10
2	Payments (count)	integer	999	
3	PaymentOptions	integer	999	1
4	Amount	decimal	999	15
5	CurrencyCode	string	999	3
6	PaymentDueDate	date	999	8
7	VariableSymbol	string	7	10
8	ConstantSymbol	string	5	4
9	SpecificSymbol	string	6	10
10	OriginatorsReferenceInformation	string	12	35
11	PaymentNote	string	1	140
12	BankAccounts (count)	integer	999	
13	IBAN	string	999	34
14	BIC	string	999	11
15	StandingOrderExt	integer	999	1
16	Day	integer	999	2
17	Month	integer	999	4
18	Periodicity	string	999	1
19	LastDate	date	999	8
20	DirectDebitExt	integer	999	1
21	DirectDebitScheme	integer	999	1
22	DirectDebitType	integer	999	1
23	VariableSymbol	string	4	10
24	SpecificSymbol	string	3	10
25	OriginatorsReferenceInformation	string	11	35
26	MandateID	string	10	35
27	CreditorID	string	9	35
28	ContractID	string	8	35
29	MaxAmount	decimal	999	15
30	ValidTillDate	date	999	8
31	BeneficiaryName	string	999	70
32	BeneficiaryAddressLine1	string	999	70
33	BeneficiaryAddressLine2	string	999	70

Appendix E - extended beneficiary fields

Three new SEPA fields were added to by square schema in version 1.1.0. In order to preserve backwards compatibility, the fields are placed at the end of the data sequence after all payments data. If these fields were put at the end of every payment, it would break backwards compatibility for bulk payments in existing readers. As a result data for more than one payment order would be corrupt.

Table 16 - PAY by square extended fields for bulk payment order

Order	Field name	Value	Field name
1	InvoiceID		
2	Payments (count)	3	
3	PaymentOptions		
...	...	...	Data for first payment
17	PaymentOptions		
...	...	...	Data for second payment
31	PaymentOptions		
...	...	...	Data for third payment
45	**BeneficiaryName**		**Belongs to the first payment**
46	**BeneficiaryAddressLine1**		**Belongs to the first payment**
47	**BeneficiaryAddressLine2**		**Belongs to the first payment**
48	**BeneficiaryName**		**Belongs to the second payment**
49	**BeneficiaryAddressLine1**		**Belongs to the second payment**
50	**BeneficiaryAddressLine2**		**Belongs to the second payment**
51	**BeneficiaryName**		**Belongs to the third payment**
52	**BeneficiaryAddressLine1**		**Belongs to the third payment**
53	**BeneficiaryAddressLine2**		**Belongs to the third payment**

References

[1] International standards ISO/IEC 18004:2006

[2] http://www.w3.org/TR/2008/REC-xml-20081126/

[3] RFC 4648 - The Base16, Base32, and Base64 Data Encodings, 2006

[4] http://www.w3.org/XML/Schema

[5] International standards ISO 4217

[6] ISO 3166-1 alpha-3

[BIC] International standards ISO 9362:2009

[IBAN] International standards ISO 13616:2007

[SEPA] http://www.ecb.int/paym/sepa/html/index.en.html