|  |  |  |
| --- | --- | --- |
|  | **LOGO**Customs Department under the Ministry of Finance of the **Republic of Lithuania** |  |
| CUSTOMS DECLARATIONS PROCESSING SYSTEMTax Free declarations QR code specificationReference iMDAS-TFQR-IS-EN Version 1.00 |

Document control

|  |  |
| --- | --- |
| Project |  |
| Product |  |
| Description | The specification defines data set of Tax Free declaration to be presented for customs control within QR code for iMDAS system by traders (sellers) application. |
| Files | pages |  |  |

Revisions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ver. | Date | Description | Actions\* | Chapters |
| 0.10 | 2021-11-03 | First draft version | C | All |
| 0.20 | 2021-11-05 | Version after LC remarks | C | All |
| 0.30 | 2021-11-08 | Version after LC remarks | C | All |
| 0.40 | 2021-11-12 | Adding customer data | C | All |
| 1.00 | 2021-11-16 | First final version | C | All |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

\* Actions: I = Insert, C = Change, D = Delete

Table of contents

[1 Introduction 3](#_Toc87269540)

[2 Input data structure 4](#_Toc87269541)

[3 QR code generation procedure 6](#_Toc87269542)

[3.1 Removing formatting from Json 6](#_Toc87269543)

[3.2 Compressing data 6](#_Toc87269544)

[3.3 Base64 encoding 6](#_Toc87269545)

[3.4 Check sum calculation 7](#_Toc87269546)

[3.5 Creating envelope structures 7](#_Toc87269547)

[3.6 QR code generation 8](#_Toc87269548)

# Introduction

Specification describes, how properly to generate QR code for Tax-Free declaration. It defines Json data structure, representing subset of Tax-Free declaration enough for Customs clearance, as well as QR code generation algorithm.

# Input data structure

QR code used for customs control (including business continuity (fallback) procedure, when Information System(s) are temporary unavailable), contains limited data set from corresponding Tax-Free declaration. Structure is Json encoded. Following table describes QR code declaration’s logical structure. Element ***docHeader*** contains header data and list of sales documents, represented by ***salesDocument*** structure. Each sales document can represent receipt or invoice. In case of receipt, ***cashRegisterReceipt*** structure should be provided. Otherwise, invoice number is required. List of goods is represented by the ***salesDocument.goods*** structure. ***salesDocument.goods.sequenceNo*** should be continuous through whole declaration. For every goods item, either ***salesDocument.goods.unitOfMeasureCode*** or ***salesDocument.goods.unitOfMeasureOther*** should be provided.

Field ***salesDocument.goods.totalAmount*** contains gross amount (incuding VAT) in EUR of the goods item.

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Type | Max Size | Remarks |
| **docHeader** |  |  |  |
| docHeader.docId | string | 34 |  |
| docHeader.docCorrNo | long | 2 |  |
| docHeader.completionDate | date | 10 | Format: YYYY-MM-DD |
| **customer** |  |  |  |
| customer.firstName | string | 200 |  |
| customer.lastName | string | 200 |  |
| **salesDocument** | list |  |  |
| **salesDocument.cashRegisterReceipt** |  |  | Either CashRegisterReceipt or InvoiceNo should be provided. |
| salesDocument.cashRegisterReceipt.cashRegisterNo | string | 50 |  |
| salesDocument.cashRegisterReceipt.receiptNo | string | 70 |  |
| salesDocument.invoiceNo | string | 70 | Either CashRegisterReceipt or InvoiceNo should be provided. |
| **salesDocument.goods** | list |  |  |
| salesDocument.goods.sequenceNo | long | 4 |  |
| salesDocument.goods.description | string | 500 |  |
| salesDocument.goods.quantity | decimal |  |  |
| salesDocument.goods.unitOfMeasureCode | string | 3 | Either UnitOfMeasureCode or UnitOfMeasureOther should be provided. |
| salesDocument.goods.unitOfMeasureOther | string | 50 |  |
| salesDocument.goods.totalAmount | decimal |  |  |

Table 1 Logical structure of the Tax-Free declaration for QR code.

Json structure for example declaration is presented below. Null elements can be omitted in real implementation.

{
 **"docHeader"**: {
 **"docId"**: **"SOME-DOC-ID"**,
 **"docCorrNo"**: 99,
 **"completionDate"**: **"2021-11-05"** },

 **"customer"**: {
 **"firstName"**: **"John"**,
 **"lastName"**: **"Doe"** },
 **"salesDocument"**: [
 {
 **"cashRegisterReceipt"**: {
 **"cashRegisterNo"**: **"15"**,
 **"receiptNo"**: **"220"** },
 **"invoiceNo"**: **null**,
 **"salesDate"**: **"2021-11-05"**,
 **"goods"**: [
 {
 **"sequenceNo"**: 1,
 **"description"**: **"Lorem ipsum dolor sit amet, consectetur adipiscing elit 1"**,
 **"quantity"**: 10,
 **"unitOfMeasureCode"**: **"NAR"**,
 **"unitOfMeasureOther"**: **null**,
 **"totalAmount"**: 56.50
 },
 {
 **"sequenceNo"**: 2,
 **"description"**: **"Lorem ipsum dolor sit amet, consectetur adipiscing elit 2"**,
 **"quantity"**: 10,
 **"unitOfMeasureCode"**: **null**,
 **"unitOfMeasureOther"**: **"Other unit of measure"**,
 **"totalAmount"**: 57.50
 }
 ]
 },
 {
 **"invoiceNo"**: **"SOME-INVOICE-NO"**,
 **"salesDate"**: **"2021-11-05"**,
 **"goods"**: [
 {
 **"sequenceNo"**: 3,
 **"description"**: **"Lorem ipsum dolor sit amet, consectetur adipiscing elit 1"**,
 **"quantity"**: 10.2,
 **"unitOfMeasureCode"**: **"KGM"**,
 **"unitOfMeasureOther"**: **null**,
 **"totalAmount"**: 56.50
 },
 {
 **"sequenceNo"**: 4,
 **"description"**: **"Lorem ipsum dolor sit amet, consectetur adipiscing elit 2"**,
 **"quantity"**: 10,
 **"unitOfMeasureCode"**: **null**,
 **"unitOfMeasureOther"**: **"Other unit of measure"**,
 **"totalAmount"**: 57.50
 }
 ]
 }
 ]
}

# QR code generation procedure

QR code has limited capacity. Theoretically it is 4 kB, however in practice, code generation software throws “data to big” exceptions little bit above 2 kB. That is why, to provide possibility of pack into QR code theoretically unlimited number of sales documents and goods items, as defined in this Specification, following procedure will be used for encoding and packing initial data structure.

## Removing formatting from Json

To maximally pack data, all formatting characters (tabs and new lines) should be removed from Json before further proceeding.

##  Compressing data

Next, Json string should be compressed by GZIP algorithm. Example Java code for compressing looks as follows.

**protected byte**[] compress(**final** String text) **throws** IOException {
 **final** ByteArrayOutputStream baos = **new** ByteArrayOutputStream();
 **final** GZIPOutputStream compressor = **new** GZIPOutputStream(baos);

 compressor.write(text.getBytes(**"utf-8"**));
 compressor.finish();
 compressor.close();
 **return** baos.toByteArray();
}

## Base64 encoding

Compressed byte array should be Base64 encoded, to place in QR code, which requires string data.

Example Java code (using org.apache.commons.codec.binary.Base64):

**byte**[] bytes = compress(json);
**return** Base64.*encodeBase64String*(bytes);

As a result, we receive String data, containing encoded binary table with zipped Json.

Example:

H4sIAAAAAAAAAK2QQWvDMAyF/4rROR1JoBvNrTSDdawNdMexg3HU1mBbqS0PRul/n92wsXU77FAcMHmSnr7nI/SkHlD26KE55p9lDw08d6v7SdstJssWiqwuyPs1QTObFaDIDgZZk2slY+quy7qaVOmr4ZTKMTDZ0W+rfeC1tLnrkfYumRn5pbSEeSBIg6ElFS06hublCEqG/QZ3OjD6DSrUA2e373KGgWqaDP3YcBbqusyO2r2RVpglF435XPGLtoAdUR/OOwMeIrpxqEqZMSifbFPKNPNEHq3QQ4hW9GTIi6BZpBRcCEUuoGLk6IXs9aCD0m4n0KSOKq04ROlY83uyLQuITnO3XaEM0eOC+ky0nm/gotLxPr/gSM/E0swtxfw809ubaXkqfgLX1wKu/wM8Uv3FC+db5JKgrbBjES4j3OUIr+l8AC8rbHt/AgAA

## Check sum calculation

Due to the limitation of single QR code, larger encoded strings will be split into several QR codes. In order to ensure, that final result is complete and not corrupted, we should calculate checksum before chunking it.

For calculation of the check sum, Adler32 algorithm should be used (Java implementation: java.util.zip.Adler32).

Example Java code:

**public** String calculateCheckSum(String data) {
 **final** Adler32 adler32 = **new** Adler32();
 adler32.update(data.getBytes());
 **return** String.*format*(**"%08X"**, adler32.getValue());
}

## Creating envelope structures

Taking into account limitation of QR code generation software and limits of precise printouts, Base64 encoded string should be split into 2 kB chunks, if it is larger than 2kB. Each chunk should be encoded in the separate QR code. In order to restore original string from chunks, every chunk should be packed into Json envelope. Example envelope is shown below. Data in the example is truncated for presentation purposes.

{
 **"data"**: **"H4sIAAAAAAAAAK2QT2vDMAzFv4r[...]"**,
 **"chunk"**: 1,
 **"total"**: 1,
 **"checksum"**: **"B24592D7"**}

Element ***data*** contains chunk of string. Element ***chunk*** informs on number of chunk, starting from 1. Element ***total*** inform about total number of chunks (and of course generated QR codes). Element ***checksum*** contains calculated checksum.

## QR code generation

QR code is generated directly from envelope structure. Example Java code for generation, using QRGen library (<https://github.com/kenglxn/QRGen>):

**public byte**[] generateQrCode(String txt) {
 txt = txt.replaceAll(**"\n"**, **""**);
 txt = txt.replaceAll(**"\r"**, **""**);
 ByteArrayOutputStream stream = QRCode.*from*(txt)
 .withErrorCorrection(ErrorCorrectionLevel.***M***)
 .withSize(400, 400)
 .stream();
 **byte**[] qrCode = stream.toByteArray();
 **return** qrCode;
}

Resulting byte array contains png image of generated QR code. It can be saved as a file or used as a dynamic resource (e.g. mobile devises) or printed on paper Tax-Free declaration..

Example image:



Picture 1 Example generated QR code