The Barcode XML format

Table of contents

1 Introduction 2

2 General structure 2

2.1 General structure in XSLT Stylesheet 2

3 Datatypes 2

4 Barcode XML Format: Abbreviated version 3

5 Human-readable message pattern 4

6 Configuration details for all supported symbologies 5

1 Introduction

This page describes the XML format used by **Barcode4J** to define barcodes. It also contains notes about the message format supported by each symbology.

2 General structure

The general structure of the Barcode XML format looks like this:

```
<barcode message="123456789">
    <{type}>
      {barcode-specific configuration}
    </{type}>
</barcode>
```

Replace **{type}** with the name of the barcode type, ex. "ean13" or "code39". You can also use the fully qualified classname of the implementing class, ex. "org.krysalis.barcode4j.impl.Code39". If you use a name it will be resolved internally using the `BarcodeClassResolver` interface.

2.1 General structure in XSLT Stylesheet

When implementing with Usage with Xalan-J, Usage with SAXON, or Usage with FOP, the general structure of the Barcode XML format changes slightly, to include the barcode namespace. For example, when using Barcode 4J with FOP, it looks like this:

```
<barcode:barcode
    xmlns:barcode="http://barcode4j.krysalis.org/ns"
    message="123456789">
  <barcode:{type}>
    {barcode-specific configuration}
  </barcode:{type}>
</barcode:barcode>
```

The barcode-specific configuration is a set of elements and attributes defined by the barcode implementation itself. Please see further down for specifics. But here's an example:

```
<barcode>
  <intl2of5>
    <height>10mm</height>
    <wide-factor>2.5</wide-factor>
  </intl2of5>
</barcode>
```

> **Note:**
> In the examples below, the sample code `<height>{length}</height>` would be changed to `<height>10mm</height>` (replacing everything between the "{" and "}" **including the "{" and "}" marks**), as is shown in the example code fragment below.

3 Datatypes

The values for the configuration elements are of various datatypes:

- {boolean}: The values `true`|`yes`|`on` or `false`|`no`|`off`.

- {string}: A simple unicode string of text like "`Hello123`".
- {char}: A unicode character.
- {number}: A number such as "2.43" or "5".
- {unit}: length unit, one of `mm`, `cm`, or `pt`.
  - The default unit is normally "mm" (Millimeters) but is implementation-dependent.
  - A special unit supported in certain places is "mw" (module width). This is used in certain 1D barcode to specify values like "10 times the module width", ex. for the quiet zone.
- {length}: A length in the form "{number}[[ ]{unit}]". Examples: `1.2mm`, `5.0`, `3`, or `10mw`.
- {checksum-mode}: The checksum mode determines the behaviour for checksums. Possible values (not all values are supported by all barcode formats):
  - add: the checksum is automatically added to the message.
  - check: the checksum is checked while rendering the barcode.
  - ignore: no checksum processing is done.
  - auto: Enables the barcode type's default behaviour.
- {human-readable-placement}: This enumeration determines the placement of the human-readable part of a barcode. Possible values:
  - bottom: the message is placed at the bottom of the symbol.
  - top: the message is placed at the top of the symbol.
  - none: no human-readable part is generated.
- {baseline-alignment}: This enumeration determines the alignment of the lines of barcodes such as POSTNET.
  - bottom: Short and long lines are aligned at the bottom.
  - top: Short and long lines are aligned at the top.
- {font-name}: As the name says, a font name used in outputting `<font-name/>`. Note that not all fonts are available for all output formats.
- {shape}: This enumeration determines the choice of symbol settings for DataMatrix barcodes.
  - force-none: Both square and rectangular symbols are allowed.
  - force-square: Force the use of square symbols.
  - force-rectangle: Force the use of rectangular symbols.
- {dimension}: Specifies a dimension (in pixels/units) in x- and y-direction. Example: "12x8" defines a size of 12 units horizontally and 8 units vertically. If you specify just one number, it is used for both directions, i.e. "12" is the same as "12x12".

Datatypes below are often given in the format "{length:15mm}". The value after the datatype indicates the default value.

> **Note:**
> If the datatype specifies a default value (as in `{length:15mm}` above), this means that the whole element is optional. In this case, the default `15mm` would be applied.

4 Barcode XML Format: Abbreviated version

The Barcode XML format can be "abbreviated" (thereby accepting all default values) as in this Code 39 example:

```
<barcode message="123456789">
```

```
  <code39/>
</barcode>
```

In this case, the defaults are assumed, and the result is effectively:

```
<barcode message="123456789">
  <code39>
    <height>15mm</height>
    <module-width>0.19mm</module-width>
    <wide-factor>2.5</wide-factor>
    <interchar-gap-width>1mw</interchar-gap-width>
    <quiet-zone enabled="true">10mw</quiet-zone>
    <checksum>auto</checksum>
    <human-readable>
      <placement>bottom</placement>
      <font-name>Helvetica</font-name>
      <font-size>8pt</font-size>
      <pattern>{string}</pattern> <!-- see below for details -->
      <display-start-stop>false</display-start-stop>
      <display-checksum>false</display-checksum>
    </human-readable>
  </code39>
</barcode>
```

5 Human-readable message pattern

General 1D symbologies have no inner structure. That's why they are used where custom messages should be encoded - most often some very big numbers with a lot of digits. These are very hard to read if there is no grouping. Or you can have a date and time encoded as contiguous set of digits. In order to make it easy for people to read complicated long messages a pattern is required. There is a facility to add custom patterns to a human-readable message. However, it only applies to Interleaves 2 of 5, Code 39, Code 128 and Codabar. Other symbologies either have no human-readable part or require special formatting of the message (like UPC/EAN).

```
<human-readable>
  <pattern>{string}</pattern>
</human-readable>
```

In the pattern the underscore "_" is used as placeholder for the next message character. The other characters within the pattern are inserted between the message symbols. The backslash "\" is escape symbol. Combination of '\?' will include '?' in the human-readable message, where '?' can be any character including underscore '_' and backslash '\'. '#' can be used to skip (delete) a character from the original message.

Examples:

Barcode message	1234567890	01012001103854	1234567890	2008-11-28
Human-readable pattern	____ ____ __	__/__/____ __:__:__ UTC	___________	____#/__#/__
Human-readable message	1234 5678 90	01/01/2001 10:38:54 UTC	1234\567_890	2008/11/28

6 Configuration details for all supported symbologies

- General 1D symbologies:
  - Interleaved 2 of 5
  - Codabar
  - Code 39
  - Code 128
- UPC/EAN group:
  - UPC-A
  - UPC-E
  - EAN-13
  - EAN-8
  - EAN-128
- Postal symbologies:
  - POSTNET
  - Royal Mail Customer Barcode (4-State)
  - USPS Intelligent Mail (4-State Customer Barcode)
- 2D symbologies:
  - PDF417
  - DataMatrix