J4L Barcodes for the Java[TM] Platform, User Guide

Contents

Introduction

J4L Barcode (also referenced as Rbarcode in this document) is a java package that creates 1D and 2D barcodes. The package can be used in several situations:

- In Java applications
- In web applications, as servlet or JSP pages
- In non Java web application as applet
- As part of Apache FOP to create PDF files
- As part of Jasper Reports
- As a web service
- In a several web application frameworks

Java code

The Java classes can be used to:

- display barcodes in your forms or panels
- create image files (png, jpeg, bmp, wbmp , gif or svg)
- or paint the barcodes in external Graphic objects

How to use the the jars as APPLET

Starting with Java version 1.7 update 45, new security restrictions has been added to Applets. This means in order to run our local test Applets you will have to temporary set the Security level to medium in the Java control panel.

When you deploy the applets in your web server you will have to :

- create a manifest for our jar file
- add a digital signature to the applet

Edit the manifest of the jar file

Open the jar file, for example with winzip and extract the file:

```
META-INF/MANIFEST.MF
```

Add the following lines to the file:

```
Permissions: sandbox
Codebase: <YourWebSiteURL>
```

And add the file to the jar file again.

Add a digital signature to the applet

If you do not have a certificate you can create one with the java keytool.exe:

- keytool.exe -genkey -keystore myKeyStore -alias mycertificate
- keytool.exe -selfcert -keystore myKeyStore -alias mycertificate -validity 3650

Once you have a certificate you can sign the jar file:

- jarsigner -keystore testkeystore barcode.jar mycertificate

Use in frames and panels

Since the barcode classes are a subclass from *java.awt.Canvas* you can add them to any container like panels. The following example is a very simple java form that displays a barcode:

```
import javax.swing.JFrame;
import com.java4less.rbarcode.BarCode;

public class BarcodeForm extends JFrame {
BarCode bc = new BarCode();

public BarcodeForm() {
this.getContentPane().setLayout(null);
bc.setBounds(new Rectangle(10, 10, 328, 253));
this.getContentPane().add(bc, null);
this.setSize( 348, 273);

// settings for the barcode
bc.code="1234";
bc.checkCharacter=true;
}

public static void main(String[] args) {
BarcodeForm barcodeForm = new BarcodeForm();
barcodeForm.show();
}

}
```

How to create a gif, png or jpg files (Java versions prior 1.4).

You can also export the barcode to a gif,png or jpeg file. In order to do this you must use the following code:

```
import com.java4less.rbarcode.*;

bc=new BarCode();
bc.setSize(400,200); // important, set size
....
new BarCodeEncoder(bc,"GIF","file.gif");
new BarCodeEncoder(bc,"PNG","file.png");
new BarCodeEncoder(bc,"JPEG","file.jpg");
```

note that:

- If you want to create gif files, the gif encoder must previously be downloaded and included in your classpath: http://www.acme.com
- If you want to create png files, the png encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar
- In order to use the JPEG encoder you just need Java 1.2 or later however gif or png produce clearer images.

How to create an image file (Java versions after 1.4).

With java 1.4 or later you have more options to create image files without addition plugins:

- PNG files: this is the recommended format
- GIF files: this may require Java version 1.6 or later
- JPG Files. This format is not recommended since in may create blurred barcodes
- BMP and WBMP files.
- SGV files: This option requires the Apache Batik files (see below)

How to created image files

Image files can be created like this:

```
import com.java4less.rbarcode.common.BarcodeImageEncoder;


BarcodeImageEncoder enc=new BarcodeImageEncoder();
enc.export("PNG", "barcode.png", barcodeObject);
```

Allowed format are PNG (recommended), JPG, GIF, BMP, and WBMP.

The barcodeObject can be any of our barcoding components (barcode , datamatrix, qrcode …)

How to create SVG files

SVG files can be created like this:

```
import com.java4less.rbarcode.common.BarcodeSVGEncoder;


BarcodeSVGEncoder enc=new BarcodeSVGEncoder();
enc.export("barcode.svg", barcodeObject);
```

note you will need to add the Apache Batik file (batik-all-1.6.jar or batik-all-1.7.jar)

How to paint on external Graphics objects

The following code illustrates how you can create a barcode in a java.awt.Image object:

```
bc=new BarCode();
bc.setSize(400,200); // important, set size

// create image
java.awt.image.BufferedImage image = new java.awt.image.BufferedImage(
bc.getSize().width,bc.getSize().height,java.awt.image.BufferedImage.TYPE_BYTE_I
NDEXED );

// get graphic context of image
java.awt.Graphics imgGraphics = image.createGraphics();

// paint barcode in graphics context of image
bc.paint(imgGraphics );
```

Servlet and JSP

The class *com.java4less.rbarcode.RBarCodeServlet* will allow you to use RBarcode as Servlet without any Java programming. In the case of servlets, the barcodes are created in the server and the output in PNG, GIF or JPEG format is sent to the browser. You can very easily use RBarCodeServlet. The servler has 2 types of parameters:

- Service parameters: these are parameters required by the servlet:
  - WIDTH: width in pixels of the output image. (default is 500).
  - HEIGHT: height in pixels of the output image. (default is 500).
  - FORMAT: format of output image, "gif", "png" or "jpeg". (default is JPEG).
- Data parameters. These are the parameters required to create the barcode. The parameters depend on the type of barcode you want to create, please look at the section Parameters in RBarcode1D, PDF417 and Datamatrix. There are however 2 common and very important parametes:
  - BARCODE: this is the value that should ne encoded
  - CODE_TYPE: used to select the barcoding symbology

note that:

- If you want to create gif files, the gif encoder must previously be downloaded and included in your classpath: http://www.acme.com
- If you want to create png files, the png encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar

Example; how to set up the servlet in Tomcat 5 (the setup is almost the same in any server):
In order to run the servlet using Tomcat 5 you can follow these steps:

1. create a new empty directory for rbarcode in Tomcat: "tomcatdirectory\webapps\rbarcode"
2. copy the content of the directory "example_servlet_jsp/war" to "tomcatdirectory\webapps\rbarcode"
3. start Tomcat
4. Open the following URL in you browser:

   http://localhost:8080/rbarcode/BarcodeServlet?BARCODE=123456789012&CHECK_CHAR=Y&CODE_TYPE=EAN13

   or if you want to change the size or format of the image add the service parameters:

   http://localhost:8080/rbarcode/BarcodeServlet?BARCODE=123456789012&WIDTH=50&HEIGHT=50&CHECK_CHAR=Y&CODE_TYPE=EAN13&CHECK_CHAR=Y&FORMAT=gif&WIDTH=600&HEIGHT=600

Note: if you want to use the PNG or the GIF encoder you must copy the jar file to "tomcatdirectory\webapps\rbarcode\WEB-INF\lib"

If you have any problems running the servlet check that the servlet is actually beeing executed, look in the java console or in the servlets logs if there are messages written by the servlet.

In most situations you will use the servlet link inside the IMG tag. In this way you can embed the image in your HTML page. For example:

```
<HTML>
<BODY bgColor=#ffffff> This is your Barcode:
<IMG height=500 width=500 src="
http://localhost:8080/rbarcode/BarcodeServlet?BARCODE=123456789012&CHECK_
CHAR=Y&CODE_TYPE=EAN13" > </BODY>
</HTML>
```

Note that you can use the size attributes *height* and *width* in order to change the size of the displayed images and also increase the printing resolution. Browsers normally print at 96 dpi, however if you need to print using a higher resolution you can do this:

1. create an image which is , for example 300 x 300 (using the HEIGH and WIDTH parameters of the servlet)
2. use the IMG tag and set the attributes height=150 and width=150
3. When you print the web page, the image will be 300 x 300 but it will take the space of a 150 x 150 image. In other words , you will be printing at about 180 dpi (instead of 96 dpi).

The product can also be used in JSP environment. The following JSP page creates the HTML output for displaying the barcode using the servlet:

```
<%String encode="jpeg";
String value="123456";
String type="BAR39";
String
servletUrl="BarcodeServlet?BARCODE="+value+"&CHECK_CHAR=Y
&CODE_TYPE="+type;response.setContentType("text/html");
response.setDateHeader ("Expires",0);

// get output stream
java.io.PrintWriter outb=response.getWriter();
// generate html
outb.print("<HTML>");
outb.print("<BODY bgColor=#ffffff>");
outb.print("This is your Barcode:<br><br>");
outb.print("<IMG src="'+servletUrl+"' >");
outb.print("</BODY>");
outb.print("</HTML>");
%>
```

If you do not want to use the servlet you can also create a JSP page that generates the barcode image like this:

```
<%
int w=400;
int h=200;response.setContentType("image/jpeg");
response.setDateHeader ("Expires",0);
response.setHeader ("Pragma", "no-cache");
response.setHeader ("Cache-Control", "no-cache");
response.setDateHeader ("Expires",0);

// get output
javax.servlet.ServletOutputStream outb=response.getOutputStream();
//create image and encoder
com.sun.image.codec.jpeg.JPEGImageEncoder encoder =
com.sun.image.codec.jpeg.JPEGCodec.createJPEGEncoder(outb );
java.awt.image.BufferedImage barImage=new
java.awt.image.BufferedImage(w,h,java.awt.image.BufferedImage.TYP
E_INT_RGB);
java.awt.Graphics2D barGraphics=barImage.createGraphics();

// create barcode
com.java4less.rbarcode.BarCode bc=new
com.java4less.rbarcode.BarCode();
bc.setSize(w,h);
bc.barType=com.java4less.rbarcode.BarCode.BAR39;
bc.code="123456";
bc.checkCharacter=true;

// work with pixels
bc.X=1;
bc.resolution=1;
```

```
bc.topMarginCM=30;
bc.leftMarginCM=30;
bc.setSize(w,h);

// paint and encode
bc.paint(barGraphics);
encoder.encode( barImage );
%>
```

Note that is you use the servlet in a Linux or Unix computer you might need to enabled the **Java Headless mode** if you get X11 errors because of a missing display.

For Java 1.5 you would use the BarcodeImageEncoder:

```
<%
int w=400;
int h=200;response.setContentType("image/png");
response.setDateHeader ("Expires",0);
response.setHeader ("Pragma", "no-cache");
response.setHeader ("Cache-Control", "no-cache");
response.setDateHeader ("Expires",0);

// get output
javax.servlet.ServletOutputStream outb=response.getOutputStream();

// create barcode
com.java4less.rbarcode.BarCode bc=new com.java4less.rbarcode.BarCode();
bc.setSize(w,h);
bc.barType=com.java4less.rbarcode.BarCode.BAR39;
bc.code="123456";
bc.checkCharacter=true;

// work with pixels
bc.X=1;
bc.resolution=1;
bc.topMarginCM=30;
bc.leftMarginCM=30;
bc.setSize(w,h);

// paint and encode
BarcodeImageEncoder enc=new BarcodeImageEncoder();
enc.export("PNG", outb, bc);
%>
```

Applet

Applets are small java programs that run inside your browser, this means they must be first downloaded from the server so that they can be executed by your browser.In order to run applets your browser must be Java enabled.

Applets are downloaded and executed when the brower finds the <APPLET> tag inside the HTML code. The tag has the following structure:

```
<APPLET
CODE=...
CODEBASE=..
ARCHIVE=.
NAME=...
WIDTH=...
HEIGHT
>
<PARAM NAME = "appletParameter1" VALUE = "value1">
<PARAM NAME = "appletParameter2" VALUE = "value2">
<PARAM NAME = "appletParameter3" VALUE = "value3">
.....
< /APPLET>
```

The green code is used to configurate the applet (size, name, Java class to be executed), and the blue code is used to pass parameters to the applet. These are the parameters you use to define the barcode.

The values for the Applet attributes are:

- CODE: this is the Java class. It can be:
  - *com.java4less.rbarcode.BCApplet.class* for 1D barcodes
  - *com.java4less.rdatamatrix.BCAppletDM.class* for Datamatrix
  - *com.java4less.rbarcode.BCApplet2D.class* for PDF 417
- ARCHIVE: it must be rbarcode.jar
- CODEBASE: this is the directory where rbarcode.jar can be found

The Barcdode 1D applet can be executed with as litte code as:

```
<HTML>
<BODY>
<APPLET
CODEBASE = "./"
CODE = "com.java4less.rbarcode.BCApplet.class"
NAME = "TestApplet"
ARCHIVE = "rbarcode.jar"
WIDTH = 500
HEIGHT = 500
ALIGN = middle
>
```

```
<PARAM NAME = "BARCODE" VALUE = "123459">
<PARAM NAME = "TYPE_CODE" VALUE = "BAR39">
</APPLET>
</BODY>
</HTML>
```

In order to run this applet you must:

1. copy this code to a file: Applet.html
2. copy rbarcode.jar to the same directory
3. Open Applet.html with your browser

Some parameters of the applet have a special format:

- Colors: valid values are: RED,BLUE,GREEN,BLACK,GRAY,LIGHTGRAY,WHITE,DARKGRAY,YELLOW,ORANGE,CYAN and MAGENTA. You can also use the RGB numerical value of a color as parameter (e.g. 0x00FF00 if green).
- Fonts have the format <font name>|<style>|<size>. Style can be PLAIN, ITALIC or BOLD. Example: "Arial|BOLD|12"

Apart from the BARCODE and CODE_TYPE parameters, there are many other you can use for the configuration of the barcode. The parameters depend on the type of barcode you want to create, please look at the section Parameters in RBarcode1D, PDF417 and Datamatrix.

You can provide the parameters in the Applet PARAM tag or you can also do it from **Javascript**. For example, the following code set a new value for the barcode:

```
TestApplet.setParameter(BARCODE,"new value");
TestApplet.refresh()
```

J4L Barcodes 1D for the Java[TM] Platform

Introduction

The following is a short description of some of the supported 1D barcode symbologies:

- **BAR39:** Code 39 is an alphanumeric bar code that can encode decimal numbers, the upper case alphabet, and the following special symbols: _ . * $ / % +. If the CHECK_CHAR flag is set RBarCode will calculate the optional check character (modulus 43).
- **BAR39EXT**: Extended Code 39 encodes the full 128 character ASCII character. If the CHECK_CHAR flag is set RBarCode will calculate the optional check character (modulus 43).
- **INTERLEAVED25**: Interleaved 2 of 5 code is a numeric only bar code. If the CHECK_CHAR flag is set RBarCode will calculate the optional modulus 10 check character.
- **UPCA**: UPC-A is used for marking products which are sold at retail in the USA. Version A encodes a twelve digit number. The first number encoded is the number system character, the next ten digits are the data characters, and the last digit is the check character.
- **EAN8**: EAN-8 is a shortened version of the EAN-13 code. It includes a 2 or 3 digit country code, 4 of 5 data digits (depending on the length of the country code), and a checksum digit.
- **EAN13**: The symbol encodes 13 characters: the first two or three are a country code which identify the country in which the manufacturer is registered (not necessarily where the product is actually made). The country code is followed by 9 or 10 data digits (depending on the length of the country code), and a checksum digit.
- **EAN128**. It is a special type of CODE128 barcode which starts with a FNC1 character.

  The initial FNC1 character will be added automatically but you can add additional FNC1 character (separators) in the barcode by using a space " " or a character 202.

- **UPCE**: The UPC-E code is a compressed barcode which is intended for use on small items. Compression works by squeezing extra zeroes out of the barcode and then automatically re-inserting them at the scanner. Only barcodes containing zeroes are candidates for the UPC-E symbol.
- **CODE128**: Code 128 is a continuous, multilevel, full ASCII code. If the CHECK_CHAR flag is set RBarCode will calculate the mandatory check character (modulus 103).

  Switchs between character sets A (upper case letters, digits and punctuation characters), B (digits, upper case and lower case characters) and C (numeric) will happen automatically in order to

minimize the length of the barcode. However you can force a character switch using the following characters in the input data: character 199 (for a switch to numeric character set C), character 201 (force a switch to character set A) and character 200 (force a switch to character set B).

- **MSI**: MSI Code is a numeric. If the CHECK_CHAR flag is set RBarCode will calculate the modulus 10 check character.
- **CODE11**: Code 11 is a numeric, high density code with one special character - .If the CHECK_CHAR flag is set RBarCode will calculate check character. If the value to be encoded is longer than 10 digits, a second check character will be calculated.
- **CODE93**: Code 93 is a more compact version of Code 39. It encodes exactly the same characters as Code 39, but uses 9 barcode elements per character instead of 15. If the CHECK_CHAR flag is set RBarCode will calculate the optional modulus 43 check character .
- **IND25**: Industrial 2 of 5 is a numeric-only barcode that has been in use a long time. Unlike Interleaved 2 of 5, all of the information is encoded in the bars; the spaces are fixed width and are used only to separate the bars. The code is self-checking and does not include a checksum.
- **CODABAR**: Codabar is a discrete, numeric code with special characters (- $:/.+). If the CHECK_CHAR flag is set RBarCode will calculate the optional modulus 16 check character .

Parameters and properties of the Java class

This section lists the properties of the BarCode class and the equivalent parameter for the servlet or applet. Please look at the JavaDoc files for a update list of parameters.

- **barType** (parameter CODE_TYPE in applet and servlet): this is the type of barcode to be used. Valid values are: BAR39, BAR39EXT, CODE39, CODE11, CODABAR, CODE93EXT, CODE128, MSI, IND25, MAT25, INTERLEAVED25, EAN13, EAN8, EAN128, POSTNET, UPCA and UPCE.
- **backColor** (BACK_COLOR in applet and servlet): back color of the barcode.
- **barColor** (BAR_COLOR in applet and servlet): color of the bars.
- **barHeightCM** (BAR_HEIGHT in applet and servlet): height of the bars in CM. If this value is 0, it will be calculated using H.
- **CODABARStartChar** (CODABAR_START in applet and servlet): Start character for CODABAR. Valid values are "A", "B", "C" or "D".
- **CODABARStopChar** (CODABAR_STOP in applet and servlet): Stop character for CODABAR. Valid values are "A", "B", "C" or "D".
- **code** (parameter BARCODE in applet and servlet): this is the value to be encoded.
- **code128set** (parameter CODE128_SET in applet and servlet): set of charaters to be used in code128. Valid values are : A, B or C. RBarcode will automatically select the correct encoding mode (set A,B or C) according to the input data.

  If it encounters at least 4 digits in a sequence while in mode A or B, it will witch to mode C: for example "ASDCC80123". You can however also force a

manual switch to mode C with ~d199 (note you must set processTilde=true) , for example ASDCC~d19980123.

Switching from mode C to A/B also happens automatically but you can force it (not required) with ~d201 (C to A) or ~d200 (C to B). For example "ASDCC801234~d201AAA" is the same as "ASDCC801234AAA"

- **checkCharacter** (CHECK_CHAR in applet and servlet): If true the software will calculate the check character automatically. The applet converts "Y" to true and "N" to false.
- **fontColor** (FONT_COLOR in applet and servlet): color of the font used to display the code.
- **guardBars** (GUARDS_BARS in applet and servlet): indicates if guardbars will be height than other bars. Only for EAN and UPC.
- **I** (parameter I in applet): intercharacter separator , only for BAR39. A value of 1 means that the separator will have the same length as X.
- **H** (parameter H in applet): Indicates how to calculate the height of the bars. A value of 0.5 means that the bars should be half the length of the symbol.
- **leftMarginCM** (LEFT_MARGIN in applet and servlet): left margin in CM.
- **N** (parameter N in applet): a value of 2, means that wide bars will be 2 times the width of narrow bars. The default vaue is 2.
- **postnetHeightTallBar** (POSTNET_TALL in applet and servlet): height (in CM) of PostNet's tall bar.
- **postnetHeightShortBar** (POSTNET_SHORT in applet and servlet): height (in CM) of PostNet's short bar.
- **processTilde** (PROCESS_TILDE): it true (or 'Y') , the tilde character in the input data will be processed as follows:

  ~dNNN represents the ascii character encoded by the 3 digits NNN. For example, ~d065 represents the character 'A'.

- **resolution:** (RESOLUTION) number of pixels/Cm. If you set it to 1 then you will be working with units=pixels
- **rotate** (ROTATE in applet and servlet): Indicates how the barcode should be painted (vertica, horizontal ...). Valid values are 0 (nomral), 90 (vertical),180 (inverted) and 270 (inverted vertical).
- **supHeight** (SUPPLEMENT_HEIGHT in applet and servlet): relative height of the supplement's bars (only EAN and UPC). The default (0.8) means 80% of the normal bars.
- **supSeparationCM** (SUPPLEMENT_SEPARATION in applet and servlet): separation between the code and the supplement (in CM). the default is 0.5 (only EAN and UPC).
- **textFont** (TEXT_FONT in applet and servlet): font used to display the code.
- **topMarginCM** (TOP_MARGIN in applet and servlet): top margin in CM.
- **UPCEANSupplement2** (SUPPLEMENT=2 in applet and servlet): indicates if the codes EAN and UPC will have a 2 digit's supplement.
- **UPCEANSupplement5** (SUPPLEMENT=5 in applet and servlet): indicates if the codes EAN and UPC will have a 5 digit's supplement.
- **UPCESystem** (UPCE_SYSTEM in applet and servlet): encoding system to be used for UPCE, valid values are 0 and 1.

- **X** (parameter X in applet and servlet): width in centimeters of narrow bars. The default is 0.03.

How to use the checkCharacter (CHECK_BAR) field:

If you are suppling the code with the check digit already calculated, you must set CHECK_CHAR to N (this is the default). If you want the software to calculate the checksum for you, you must set CHECK_CHAR to Y.

For EAN and UPC have fixed length and therefore you only have the following possibilities:

- EAN13: you supply a 13 digits code and set CHECK_CHAR to N or you supply a 12 digits code and set CHECK_CHAR to Y.
- EAN8: you supply a 8 digits code and set CHECK_CHAR to N or you supply a 7 digits code and set CHECK_CHAR to Y.
- UPCA and UPCE: you supply a 12 digits code and set CHECK_CHAR to N or you supply a 11digits code and set CHECK_CHAR to Y.

J4L PDF 417 and Macro PDF 417 for the Java[TM] Platform

Introduction

PDF stands for “Portable Data File.” A two-dimensional symbology (2D), a single PDF417 symbol carries up to 1.1 kilobytes of machine-readable data in a space no larger than a standard bar code. And, unlike one-dimensional bar codes (1D), which are just a key linked to a database, PDF417 symbols contain the database itself. That means, you don't have to store an article number in the barcode but you can also store the name , the size , the color, the name of the manufacturer etc...

The basic characteristics are:

- Each PDF417 symbol consists of a stack of vertically aligned rows with a minimum of 3 rows (maximum 90 rows). Each row can have 1 to 30 columns.
- Three compaction modes:
  - Text Compaction mode allows all printable ASCII characters to be encoded (i.e. values 32 to126 and some additional control characters)
  - Byte Compaction mode allows any byte values to be encoded.
  - Numeric Compaction is a more efficient mode for encoding numeric data
- The maximum capacity is (at error correction level 0):
  - Text Compaction mode: 1 850 characters
  - Byte Compaction mode: 1 108 characters
  - Numeric Compaction mode: 2 710 characters
- Macro PDF417: this feature allows large amount of data to be encoded in a secuence of linked PDF417 symbols. Up to 99 999 different PDF417 symbols can be concatenated using this mechanism.
- Compact PDF417: In clean environments, it is possible to reduce the size of the symbol by removing some non-data bars.Parameters

Parameters and properties of the Java class

This section lists the properties of the BarCode2D class and the equivalent parameter for the servlet or applet. Please look at the JavaDoc files for a update list of parameters.

Since it is an extension of RBarcode, therefore the some parameters are inherited from the BarCode class:

- CODE_TYPE: must have the value PDF417.
- BARCODE: text to encode.
- BACK_COLOR.

- BAR_COLOR
- X
- BAR_HEIGHT
- RESOLUTION

The properties specific to PDF417 are:

- **PDFColumns** (PDF_COLUMNS in applet and servlet): number of columns for PDF417 (the default is 10).
- **PDFECLevel** (PDF_ECLEVEL in applet and servlet): error correction level for PDF417 (the default is 0).
- **PDFMode** (PDF_COMPACTION in applet and servlet): PDF417 mode can be NUMERIC, TEXT or BINARY.
- **PDFRows** (PDF_ROWS in applet and servlet): number of rows for PDF417 (optional).
- **PDFMaxRows** (PDF_MAXROWS in applet and servlet): number of rows for PDF417 (optional).

Note: The Macro PDF properties are described in the JavaDoc files.

J4L Datamatrix for the Java[TM] Platform

Introduction

The package RDataMatrix contains an applet and a java class that will allow you to create data matrix (ECC200) barcodes for you java applications or HTML pages.

Data Matrix is a two-dimensional (2D) matrix symbology which is made up of square modules arranged within a perimeter finder pattern. It can encode up to 3116 characters from the entire 256 byte ASCII character set. The symbol consists of data regions which contain square modules set out in a regular array. Large ECC 200 symbols contain several regions. Each data region is delimited by a finder pattern, and this is surrounded on all four sides by a quiet zone border (margin).

ECC 200 symbols have an even number of rows and an even number of columns. Most of the symbols are square with sizes from 10 x 10 to 144 x 144. Some symbols however are rectangular with sizes from 8 x 18 to 16 x 48. All ECC 200 symbols can be recognized by the upper right corner module being light (binary 0).

ECC200 is the newest version of data matrix and supports advanced encoding error checking and correction algorithms (reed-solomon). This algorithms allow the recognition of barcodes that are up to 60% damaged.

The barcode supports two optional mechanisms:

- The "Extended Channel Interpretation" (ECI) mechanism enables characters from other character sets (e.g. Arabic, Cyrillic ..) and other data interpretations or industry-specific requirements to be represented.
- The "Structured append" allows files of data to be represented as a secuence of up to 16 Data Matrix symbols. The original data or file can be reconstructed regardless of the order of the symbols.

RDataMatrix supports:

- All sizes and formats (from 10x10 till 144x144)
- Ascii, text , C40 and Base256 (for binary data) encoding.
- The "Extended Channel Interpretation and Structured append

Formats

RDataMatrix supports all data matrix formats. The following table contains the size , the capacity and the correction error features of each format

Size	Numeric Capacity	Alphanumeric capacity	Binary capacity	Max Correctable Error/Erasure
10 x 10	6	3	1	2
12 x 12	10	6	3	3
14 x 14	16	10	6	5/7
16 x 16	24	16	10	6/9
18 x 18	36	25	16	7/11
20 x 20	44	31	20	9/15
22 x 22	60	43	28	10/17
24 x 24	72	52	34	12/21
26 x 26	88	64	42	14/25
32 x 32	124	91	60	18/33
36 x 36	172	127	84	21/39
40 x 40	228	169	112	24/45
44 x 44	288	214	142	28/53
48 x 48	348	259	172	34/65
52 x 52	408	304	202	42/78
64 x 64	560	418	278	56/106
72 x 72	736	550	366	72/132
80 x 80	912	682	454	96/180
88 x 88	1152	862	574	112/212
96 x 96	1392	1042	694	136/260
104 x 104	1632	1222	814	168/318
120 x 120	2100	1573	1048	204/390
132 x 132	2608	1954	1302	248/472
144 x 144	3116	2335	1556	310/590
8 x 18	10	6	3	3
8 x 32	20	13	8	5
12 x 26	32	22	14	7/11
12 x 36	44	31	20	9/15
16 x 36	64	46	30	12/21
16 x 48	98	72	47	14/25

Encoding

The data represented in the symbol is can be compressed using one or several of the following algorithms:

- ASCII: it is used to encode data that mainly contains ascii characters (0-127). It encodes one alphanumeric or two numeric characters per byte.
- C40: it is used to encode data that mainly contains numeric and upper case characters. C40 encodes three alphanumeric data characters into two bytes.
- TEXT: it is used to encode data that mainly contains numeric and lowercase characters. TEXT encodes three alphanumeric data characters into two bytes.
- BASE256: it is used to encode 8 bit values.

All encoding system can be used to encode any data, but for example, encoding binary data with C40 generates much more overhead (longer symbol) than with BASE256.

Control characters

RDataMatrix uses the character ~ to recognize some special characters in the input data. The following possibilities are available:

- ~*X* is used to represent character values from 0 to 26. Replace the *X* like in the following example ~@ = means character ascii 0, ~A= means character 1, ~B=means character 2, ~C=means character 3 ...
- ~1: represents the character FNC1. When FNC1 appears in the first position (or in the fifth position of the first symbol of a Structured Append), it will indicate that the data conforms to the UCC/EAN Application Identifier standard format.
- ~2: It is used to represent Structured Append. Structured Append is used to link information from several symbols in a secuence. The ~2 must be followed by 3 additional bytes. The first 4 bits of thie first byte identify the position of the particular symbol in the secuence . The last 4 bits identify the total number of symbols in the secuence. The second and third byte are used as a file identifier are can have a value between 1 and 254 (up to 254*254=64516 identifiers). See Data Matrix Specification for more information about this (ISO 16022).
- ~3: This character are only allowed in the first position of the symbol. It indicates that the data contains commands for the barcode reader.
- ~4: not allowed.
- ~5 and ~6: These characters are only allowed in the first position of the symbol. If ~5 is used the header [)> ascii*30* ascii*05* ascii*29* will be transmitted by the barcode reader before the data in the symbol and the trailer ascii*30* ascii*04* will be transmitted after the data. If a ~6 is used , the header [)> ascii*30* ascii*05* ascii*29* will be transmittedby the reader before the data and the trailer ascii*30* ascii*04* will be transmitted afterwards.
- ~7NNNNNN specifies the Extended Channel to be used, where NNNNNN is a value between and 000000 - 999999. For example: ~7000010 means Extended Channel 10 . Extended channel is used for using other character

sets other than ascii. See Data Matrix Specification for more information about this (ISO 16022).
- ~dNNN represents the ascii character encoded by the 3 digits NNN. For exmaple, ~d065 represents the character 'A'.

Parameters and properties of the Java Class

RDataMatrix is an extension of RBarcode, therefore the some parameters are inherited from the BarCode class:

- BARCODE: text to encode.
- BACK_COLOR.
- BAR_COLOR
- ROTATE
- CODE_TYPE: must have the value DATAMATRIX.

The specific parameters for this type of symbology are:

- DM_DOT_PIXELS: size of the square modules in pixels (the default is 8 pixels)
- DM_TILDE: if true ("Y") the tilde (~) will be processed as already explained. If not it will be treated as a normal character.
- DM_MARGIN_PIXELS: quite zone around the symbol (the default is 30 pixels).
- DM_ENCODING: the encoding to be used. The default is ASCII. Posible values are ASCII, C40, TEXT, BASE 256 and AUTO
- DM_FORMAT: If empyte the format will be selected automatically, if not you can specify the format (e.g. C24X24).

J4L MicroPDF417 for the Java[TM] Platform

Introduction

The package RMicroPDF417 contains the classes you need to create Micro PDF 417 barcodes within your jsp or java applications.

MicroPDF417 is a 2D barcoding symbology based on PDF417. Micro PDF 417 has been specifically designed to provide even greater space efficiency for small item marking applications than the standard PDF 417 symbology.

Micro PDF 417 has the following encoding methods:

- Text mode: can encode all printable ASCII characters i.e. values 32 to 126 inclusive.
- Binary mode: can encode all 256 possible 8-bit byte values.
- Numeric mode: can efficiently encode numeric data strings.

The maximum data capacity of micro PDf symbols is:

- Text encoding mode: 250 characters.
- Byte encoding mode: 150 characters.
- Numeric encoding mode: 366 characters.

Micro PDF 417 barcodes can have 1 to 4 columns of data and they support a predefined set of formats with fixed error correction levels. The following table shows the available formats:

Columns / Rows	Capacity (binary encoding)	Capacity (text encoding)	Capacity (numeric encoding)
FORMAT_1X11	3	6	8
FORMAT_1X14	7	12	17
FORMAT_1X17	10	18	26
FORMAT_1X20	13	22	32
FORMAT_1X24	18	30	44

FORMAT_1X28	22	38	55
FORMAT_2X8	8	14	20
FORMAT_2X11	14	24	35
FORMAT_2X14	21	36	52
FORMAT_2X17	27	46	67
FORMAT_2X20	33	56	82
FORMAT_2X23	38	64	93
FORMAT_2X26	43	72	105
FORMAT_3X6	6	10	14
FORMAT_3X8	10	18	26
FORMAT_3X10	15	26	38
FORMAT_3X12	20	34	49
FORMAT_3X15	27	46	67
FORMAT_3X20	39	66	96
FORMAT_3X26	54	90	132
FORMAT_3X32	68	114	167
FORMAT_3X38	82	138	202
FORMAT_3X44	97	162	237
FORMAT_4X4	8	14	20
FORMAT_4X6	13	22	32
FORMAT_4X8	20	34	49
FORMAT_4X10	27	46	67
FORMAT_4X12	34	58	85
FORMAT_4X15	45	76	111
FORMAT_4X20	63	106	155
FORMAT_4X26	85	142	208
FORMAT_4X32	106	178	261
FORMAT_4X38	128	214	313
FORMAT_4X44	150	250	366

RMicroPDF417 supports:

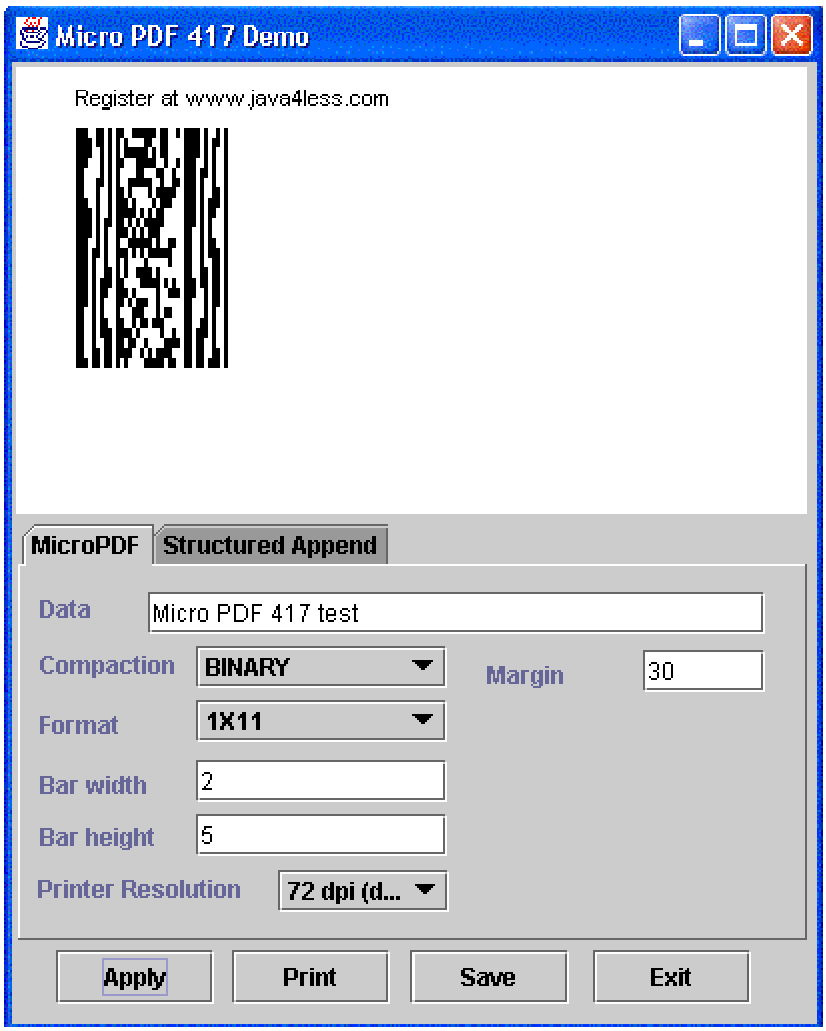
- All Micro PDF417 configurations.
- Structured Append. This version allows to concatenate several PDF417 symbols in order to encode larger amount of information.
- All encoding schemes: text, numeric and binary.

Installation

In order to install the software you just need to unzip the ZIP file in an empty directory (you must unzip with pathname so that the subdirectories are properly created.

Examples

In order to run the sample application you must execute **runSUN.bat**. The sample application requires JDK 1.3 or later.



In the sample application you can set following properties all PDF and StructuredAppend PDF properties. If Segment is not -1, a StructuredAppend pdf symbol will be created. Otherwise a normal PDF 417 symbol is created.

You can execute the following commands:

- Apply: repaint the symbol using the new properties.
- Print: send imahe to printer at the selected resolution.
- Save: save the symbol in jpg format. We recommend you to use gif or png formats,however the sample application uses JPG because it does not require you to download any additional package. In order to create gif or png files you must download an external encoder (see javadoc).
- Exit: terminate application.

The Servlet

RMicroPDF417 comes with a standard servlet (RMicroPDF417Servlet) you can use to create micro PDF images. The parameters for the servlet can be sent using GET or POST commands. The parameters are:

- DEBUG: use ON to activate verbose output.
- MODE: encoding mode (default is BINARY). Value values are:
  - BINARY: can encode any character.
  - TEXT: can encode alphanumeric characters and the following punctuation characters: ! \ # $ % & ' ( ) * + , - . / { } | ~
  - NUMERIC: encodes digit characters only
- DATA: string to be encoded.
- MODULE_WIDTH: width in pixels of the bars (default is 1).
- BAR_HEIGHT: height in pixels of the bars (default is 4).
- TOP_MARGIN: top margin in pixels (default is 20).
- LEFT_MARGIN: left margin in pixels (default is 20).
- CONFIGURATION: preferred format of the micro pdf 417 (See table for available formats, for example FORMAT_1X11).
- WIDTH: size of the image. Default is 400.
- HEIGHT: size of the image. Default is 200.
- FORMAT: output format: PNG, GIF or JPEG (default):
  - The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
  - The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar

Note: the servlet does not include parameters for the Structured Append properties. If required new parameters can be added.

For example, if you want to test the servlet using Tomcat 4.1 you must do the following:

1. copy the RMicroPDF417.jar to */tomcat/webapps/examples/Web-inf/lib*
2. copy the RMicroPDF417Servlet.class to */tomca/webappst/examples/Web-inf/classes*
3. Start the server an test the servlet entering the following address and parameters in your browser:

   http://localhost:8080/examples/servlet/RMicroPDF417Servlet?DATA=ABC&CONFIGURATION=FORMAT_1X14

In order to property print a barcode created with the servlet you must use the <IMG> tag. In this way you can embed the image in your HTML page. Note that you will need to use the attibutes height and width in order to achieve the correct size of the barcode on the printed page.

This is a simple HTML page that contains the barcode:

```
<HTML>
<HEAD>

        <TITLE>Servlet Example META http-equiv=Content-Type
        content="text/html; charset=windows-1252">

</HEAD>
<BODY bgColor=#ffffff>

        This is your Barcode:
        <IMG height=100 width=100
        src="http://localhost:8080/examples/servlet/RMicroPDF417Servle
        t?DATA=ABC&CONFIGURATION=FORMAT_1X14" >

</BODY>
</HTML>
```

J4L Aztec Code for the Java[TM] Platform

Introduction

The package RAztec contains the classes you need to create Aztec barcodes within your jsp or java applications.

Aztec Code is a 2D matrix symbology made up of square modules on a square grid, with a square bullseye pattern at their center. Aztec Code symbols can encode large amounts of data with user defined error correction level.

The smallest format can encode 13 numeric , 12 alphabetic characters or 6 bytes of data, while the largest format can encode 3832 numeric ,3067 alphabetic characters or 1914 bytes of data.

available formats are:

Rows / columns (number of modules/squares)	Capacity (digits)	Capacity (text)	Capacity (binary data)
CONFIGURATION_15X15_COMPACT	13	12	6
CONFIGURATION_19X19	18	15	8
CONFIGURATION_19X19_COMPACT	40	33	19
CONFIGURATION_23X23	49	40	24
CONFIGURATION_23X23_COMPACT	70	57	33
CONFIGURATION_27X27	84	68	40
CONFIGURATION_27X27_COMPACT	110	89	53
CONFIGURATION_31X31	128	104	62
CONFIGURATION_37X37	178	144	87
CONFIGURATION_41X41	232	187	114
CONFIGURATION_45X45	294	236	145
CONFIGURATION_49X49	362	291	179
CONFIGURATION_53X53	433	348	214
CONFIGURATION_57X57	516	414	256
CONFIGURATION_61X61	601	482	298
CONFIGURATION_67X67	691	554	343
CONFIGURATION_71X71	793	636	394
CONFIGURATION_75X75	896	896	446

CONFIGURATION_79X79	1008	808	502
CONFIGURATION_83X83	1123	900	559
CONFIGURATION_87X87	1246	998	621
CONFIGURATION_91X91	1378	1104	687
CONFIGURATION_95X95	1511	1210	753
CONFIGURATION_101X101	1653	1324	824
CONFIGURATION_105X105	1801	1442	898
CONFIGURATION_109X109	1956	1566	976
CONFIGURATION_113X113	2216	1694	1056
CONFIGURATION_117X117	2281	1826	1138
CONFIGURATION_121X121	2452	1963	1224
CONFIGURATION_125X125	2632	2107	1314
CONFIGURATION_131X131	2818	2256	1407
CONFIGURATION_135X135	3007	2407	1501
CONFIGURATION_139X139	3205	2565	1600
CONFIGURATION_143X143	3409	2728	1702
CONFIGURATION_147X147	3616	2894	1806
CONFIGURATION_151X151	3832	3067	1914

Compact formats can be used to encode short messages in a more efficient manner than full range formats. Note that reader/decoder can autodiscrimanate between both formats.

There are also a set of 256 special formats called "Aztec runes" which can be used for encoding values 0 to 255 for special applications.

RAztecCode supports:

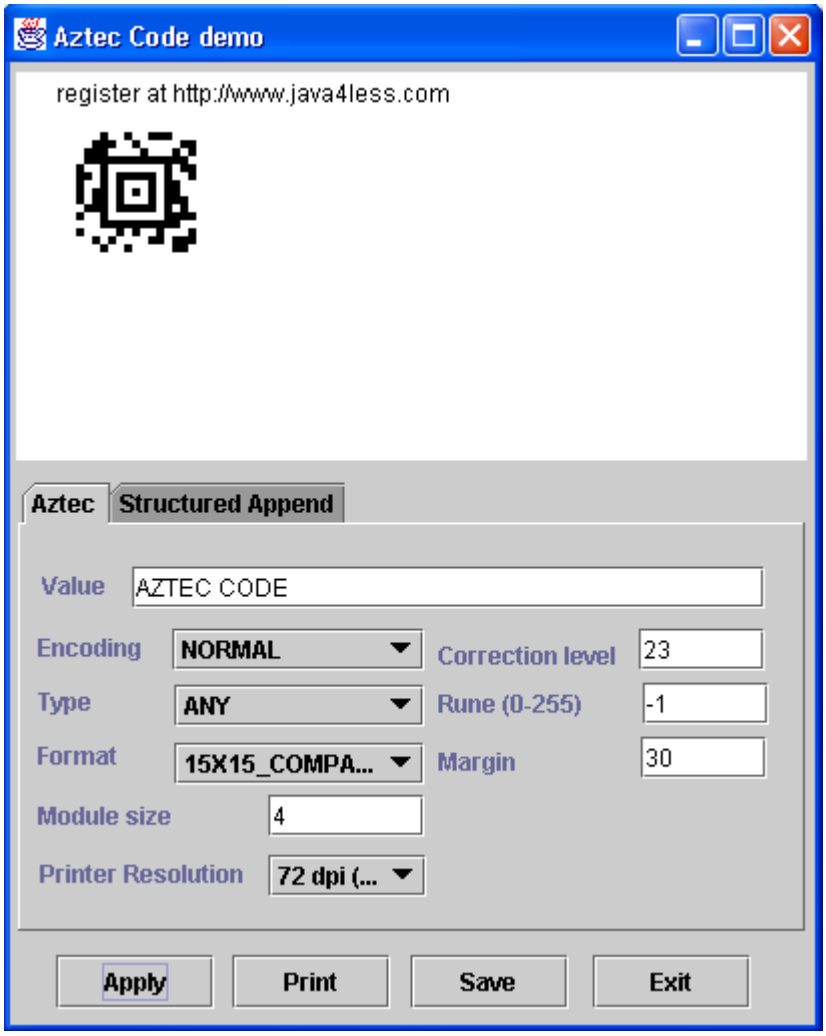
- All Aztec code formats (compact and full range).
- Normal and binary encoding.
- Structured append. This allows you to encode large messages by means of a secuence of symbols.
- Extended Channel Interpretation.
- Reader initialization bit.
- Aztec Runes.

Installation and requirements

In order to install the software you just need to unzip the ZIP file in an empty directory (your must unzip with pathname so that the subdirectories are properly created.

Examples

In order to run the sample application you must execute **runSUN.bat**. The sample application requires JDK 1.3 or later.



In the sample application you can set all properties of the Aztec code symbology.

You can execute the following commands:

- Apply: repaint the symbol using the new properties.
- Print: send imahe to printer at the selected resolution.
- Save: save the symbol in jpg format. We recommend you to use gif or png formats,however the sample application uses JPG because it does not require you to download any additional package. In order to create gif or png files you must download an external encoder (see javadoc).

- Exit: terminate application.

The Servlet

RAztec comes with a standard servlet (RAztecServlet) you can use to create aztec images. The parameters for the servlet can be sent using GET or POST commands. The parameters are:

- DEBUG: use ON to activate verbose output.
- ERROR_CORRECTION: percentage of errors which can be recovered (default is 23, 23%)
- INIT_READER: if "Y" the reader initialization bit will be enabled (default is N).
- ENCODING: encoding mode (default is NORMAL). Value values are:
  - NORMAL: can encode any character but it is not very efficient encoding binary values (values above 128).
  - BINARY: use this mode only if your data contains many bytes/characters above 128.
- PROCESSTILDE: if true (default) the tilde character (~) will be processed like this:
  - ~~: will be replaced with ~
  - ~dxxx: will be replaced by the character whose ascii code is xxx. For example ~d065 will be replaced with A.
  - ~F: will be replaced with the FNC1 flag (allowed as first codeword only).
  - ~Exxxxxx: will be replaced with the Extended Interpretation Channel flag xxxxxx. For example to activate Extended Interpretation Channel 1, use ~E000001.
- DATA: string to be encoded.
- MODULE_SIZE: number of pixels which make a module (square) in the barcode (default is 4).
- CONFIGURATION: preferred format. Another format will be automatically selected if AutoConfigurate=true and the amount of data and the selected error correction level does not fit in the preferred format. Valid values are CONFIGURATION_15X15_COMPACT, CONFIGURATION_19X19 .... and so on (see table)
- TYPE: use this property to define which formats can be used: ANY (any), COMPACT (only compact formats) or FULL (only full range formats).
- RUNE: set a value between 0 and 255 to create a Aztec code rune (default is -1, disabled)
- MARGIN: left and top margin in pixels (default is 30).
- WIDTH: size of the image. Default is 400.
- HEIGHT: size of the image. Default is 200.
- FORMAT: output format: PNG, GIF or JPEG (default):
  - The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
  - The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar

For example, if you want to test the servlet using Tomcat 4.1 you must do the following:

1. copy the RAztec.jar to */tomcat/webapps/examples/Web-inf/lib*
2. copy the RAztecServlet.class to */tomca/webappst/examples/Web-inf/classes*
3. Start the server an test the servlet entering the following address and parameters in your browser:

   http://localhost:8080/examples/servlet/RAztecServlet?DATA=ABC&ENCODING=NORMAL

In order to property print a barcode created with the servlet you must use the <IMG> tag. In this way you can embed the image in your HTML page. Note that you will need to use the attibutes height and width in order to achieve the correct size of the barcode on the printed page.

This is a simple HTML page that contains a barcode:

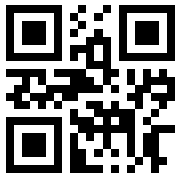
```
<HTML>
<HEAD>

        <TITLE>Servlet Example META http-equiv=Content-Type
        content="text/html; charset=windows-1252">

</HEAD>
<BODY bgColor=#ffffff>

        This is your Barcode:
        <IMG height=100 width=100
        src="http://localhost:8080/examples/servlet/RAztecServlet?DATA
        =TEST" >

</BODY>
</HTML>
```

J4L QRCode for the Java[TM] platform

Introduction

The package J4L-QRCode contains the classes you need to create QRCode barcodes within your jsp or java applications.

QR Code is a matrix symbology which includes a finder pattern located at three corners of the symbol used to locate the symbol and figure out its size and orientation.



The main features of QRCode symbols are:

- There are 40 sizes of QR Code symbols (called Version 1, Version 2 till Version 40). Version 1 measures 21 modules * 21 modules, Version 2 measures 25 modules * 25 modules and so on. Version 40 measures 177 modules * 177 modules.
- The following data can be encoded:
    - Numeric data (digits 0-9).
    - Alphanumeric characters , digits 0 - 9; upper case letters A -Z and nine other characters: space, $ % * + - . / :
    - Byte data (bytes 0-255)
    - Kanji characters (hexadecimal values 8140 -9FFC and E040 - EBBF)
- Symbol size is 21 * 21 modules to 177 * 177 modules (Versions 1 to 40, increasing in steps of 4 modules per side).
- The maximum number of characters encoded in one symbol (without structured append) is:
    - Numeric data: 7089 characters
    - Alphanumeric data: 4296 characters
    - Byte data: 2953 characters
    - Kanji data: 1817 characters
- Supports 4 error correction levels:
    - L (7% of the symbol codewords).
    - M (15% of the symbol codewords).
    - Q (25% of the symbol codewords).
    - H (30% of the symbol codewords).
- Structured append (optional) This allows files of data to be represented logically in up to 16 QR Code symbols.
- Extended Channel Interpretation (optional): enables data using character sets other than the default set (e.g. Arabic, Cyrillic, Greek).
- FNC1 indicators: FNC1 mode is used for messages containing data formatted either in accordance with the UCC/EAN Application Identifiers standard or in

accordance with a specific industry standard previously agreed with AIM International.

J4L-QRCode supports:

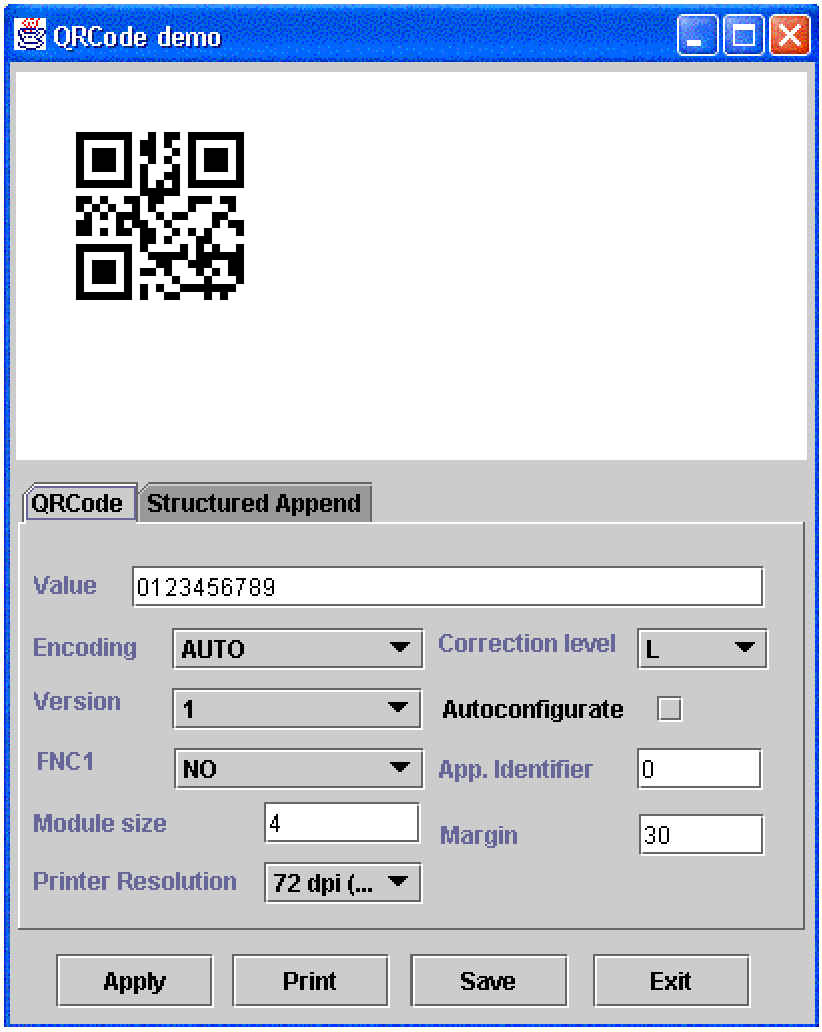
- QRCode mode 2 symbols (not mode 1)
- All versions 1-40. Automatic selection of the version is also supported.
- All encoding method (numeric, alphanumeric, byte and kanji). Automatic selection of the encoding method is also supported.
- Structured append.
- Extended Channel Interpretation (only 1 per symbol, no nesting supported).
- All 4 error correction levels.
- FNC1 indicators.

Installation and requirements

In order to install the software you just need to unzip the ZIP file in an empty directory (your must unzip with pathname so that the subdirectories are properly created.

Examples

In order to run the sample application you must execute **runSUN.bat**. The sample application requires JDK 1.3 or later.



In the sample application you can set all properties of the QRCode symbology.

You can execute the following commands:

- Apply: repaint the symbol using the new properties.
- Print: send image to printer at the selected resolution.
- Save: save the symbol in jpg format. We recommend you to use gif or png formats,however the sample application uses JPG because it does not require you to download any additional package. In order to create gif or png files you must download an external encoder (see javadoc).

- Exit: terminate application.

The Servlet

QRCode comes with a standard servlet (QRCodeServlet) you can use to create qrcode images. The parameters for the servlet can be sent using GET or POST commands. The parameters are:

- **APPLICATION_INDICATOR:** see *Fnc1Mode* property.
- **AUTOCONFIGURATE:** if true (value Y) the preferredVersion can be ignored if the data does not fit in the selected version.
- **BAR_COLOR**: color of the bars. You can also use the RGB numeric value of a color as parameter. For example "0xFF0000" is red (RGB format), "00x00FF00" is green and so on.
- **BACK_COLOR**: color of the background.
- **DEBUG**: use ON to activate verbose output.
- **ERROR_CORRECTION**: valid values are L, M , Q , H .
- **ENCODING**: encoding mode (default is AUTO). Valid values are:
  - AUTO: Automatic selecting of the encoding method.
  - ALPHA: encode alpahnumeric characters only (upper case letter plus 9 additional characters).
  - NUMERIC: encode alpahnumeric digits only.
  - BYTE: use this mode to encode binary data.
  - KANJI: encodes Kanji characters only.
- **FNC1_MODE**: Selects FNC1 mode. Valid values are:
  - FNC1_MODE_NO: disable FNC1 (default).
  - FNC1_MODE_FIRST: This Mode Indicator identifies symbols encoding data formatted according to the UCC/EAN Application Identifiers standard.
  - FNC1_MODE_SECOND: This Mode Indicator identifies symbols formatted in accordance with specific industry or application specifications previously agreed with AIM International. You must then set a value for the *ApplicationIndicator* property.
- **PROCESSTILDE**: if true (value Y) the tilde character (~) will be processed like this:
  - ~~: will be replaced with ~
  - ~dxxx: will be replaced by the character whose ascii code is xxx. For example ~d065 will be replaced with A.
- **DATA**: string to be encoded.
- **MARGIN**: left and top margin in pixels (default is 30).
- **MODULE_SIZE**: number of pixels which make a module (square) in the barcode (default is 4).
- **WIDTH**: size of the image. Default is 400.
- **HEIGHT**: size of the image. Default is 200.
- **FORMAT:** output format: PNG, GIF or JPEG (default):

    - The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
    - The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar
- **STRUCTURED_APPEND:** if true (value Y), the structured append mode is enabled (default is false).
- **STRUCTURED_APPEND_COUNTER:** number of symbols which make the sequence.
- **STRUCTURED_APPEND_INDEX:** position of current symbol within the secuence (starting at 1).
- **CONFIGURATION**: preferred format (version). Another version will be automatically selected if AutoConfigurate=true and the amount of data and the selected error correction level does not fit in the preferred version. Valid values are 1 to 40.

For example, if you want to test the servlet using Tomcat 4.1 you must do the following:

1. copy the qrcode.jar to */tomcat/webapps/examples/Web-inf/lib*
2. Start the server and test the servlet entering the following address and parameters in your browser:

   http://localhost:8080/examples/servlet/QRCodeServlet?DATA=ABC&ENCODING=NORMAL

In order to property print a barcode created with the servlet you must use the <IMG> tag. In this way you can embed the image in your HTML page. Note that you will need to use the attibutes height and width in order to achieve the correct size of the barcode on the printed page.

This is a simple HTML page that contains a qrcode symbol:

```
<HTML>
<HEAD>

        <TITLE>Servlet Example META http-equiv=Content-Type
        content="text/html; charset=windows-1252">

</HEAD>
<BODY bgColor=#ffffff>

        This is your Barcode:
        <IMG height=100 width=100
        src="http://localhost:8080/examples/servlet/QRCodeServlet?DATA=TE
        ST" >

</BODY>
</HTML>
```

The Applet

You must add the following code in your html pages to use the applet:

```
<APPLET
CODEBASE = "./"
CODE =
"com.java4less.qrcode.QRCodeApplet.class"
ARCHIVE = "qrcode.jar"
NAME = "TestApplet"
WIDTH = 300
HEIGHT = 250
HSPACE = 0
VSPACE = 0
ALIGN = middle
>

<PARAM NAME = "DATA" VALUE = "123411">

</APPLET>
```

Note that the qrcode.jar file must be located in the directory referenced by the CODEBASE parameter.

The available paramete are the same as those used by the servlet (see description in the previous section):

- **APPLICATION_INDICATOR**
- **AUTOCONFIGURATE**
- **BAR_COLOR**
- **BACK_COLOR**
- **ERROR_CORRECTION**
- **ENCODING**
- **FNC1_MODE**
- **PROCESSTILDE**
- **DATA**
- **MARGIN**
- **MODULE_SIZE**
- **STRUCTURED_APPEND**
- **STRUCTURED_APPEND_COUNTER**
- **STRUCTURED_APPEND_INDEX**
- **CONFIGURATION**

Annex A

Table — Data capacity for QRCode versions

Version	Error Correction Level	Numeric	Alphanumeric	Byte	Kanji
1	L	41	25	17	10
	M	34	20	14	8
	Q	27	16	11	7
	H	17	10	7	4
2	L	77	47	32	20
	M	63	38	26	16
	Q	48	29	20	12
	H	34	20	14	8
3	L	127	77	53	32
	M	101	61	42	26
	Q	77	47	32	20
	H	58	35	24	15
4	L	187	114	78	48
	M	149	90	62	38
	Q	111	67	46	28
	H	82	50	34	21
5	L	255	154	106	65
	M	202	122	84	52
	Q	144	87	60	37
	H	106	64	44	27
6	L	322	195	134	82
	M	255	154	106	65
	Q	178	108	74	45
	H	139	84	58	36
7	L	370	224	154	95
	M	293	178	122	75
	Q	207	125	86	53
	H	154	93	64	39
8	L	461	279	192	118
	M	365	221	152	93
	Q	259	157	108	66
	H	202	122	84	52
9	L	552	335	230	141
	M	432	262	180	111
	Q	312	189	130	80
	H	235	143	98	60
10	L	652	395	271	167
	M	513	311	213	131
	Q	364	221	151	93
	H	288	174	119	74
11	L	772	468	321	198
	M	604	366	251	155

	Q	427	259	177	109
	H	331	200	137	85
12	L	883	535	367	226
	M	691	419	287	177
	Q	489	296	203	125
	H	374	227	155	96
13	L	1022	619	425	262
	M	796	483	331	204
	Q	580	352	241	149
	H	427	259	177	109
14	L	1101	667	458	282
	M	871	528	362	223
	Q	621	376	258	159
	H	468	283	194	120
15	L	1250	758	520	320
	M	991	600	412	254
	Q	703	426	292	180
	H	530	321	220	136
16	L	1408	854	586	361
	M	1082	656	450	277
	Q	775	470	322	198
	H	602	365	250	154
17	L	1548	938	644	397
	M	1212	734	504	310
	Q	876	531	364	224
	H	674	408	280	173
18	L	1725	1046	718	442
	M	1346	816	560	345
	Q	948	574	394	243
	H	746	452	310	191
19	L	1903	1153	792	488
	M	1500	909	624	384
	Q	1063	644	442	272
	H	813	493	338	208
20	L	2061	1249	858	528
	M	1600	970	666	410
	Q	1159	702	482	297
	H	919	557	382	235
21	L	2232	1352	929	572
	M	1708	1035	711	438
	Q	1224	742	509	314
	H	969	587	403	248
22	L	2409	1460	1003	618
	M	1872	1134	779	480
	Q	1358	823	565	348
	H	1056	640	439	270
23	L	2620	1588	1091	672
	M	2059	1248	857	528
	Q	1468	890	611	376

	H	1108	672	461	284
24	L	2812	1704	1171	721
	M	2188	1326	911	561
	Q	1588	963	661	407
	H	1228	744	511	315
25	L	3057	1853	1273	784
	M	2395	1451	997	614
	Q	1718	1041	715	440
	H	1286	779	535	330
26	L	3283	1990	1367	842
	M	2544	1542	1059	652
	Q	1804	1094	751	462
	H	1425	864	593	365
27	L	3517	2132	1465	902
	M	2701	1637	1125	692
	Q	1933	1172	805	496
	H	1501	910	625	385
28	L	3669	2223	1528	940
	M	2857	1732	1190	732
	Q	2085	1263	868	534
	H	1581	958	658	405
29	L	3909	2369	1628	1002
	M	3035	1839	1264	778
	Q	2181	1322	908	559
	H	1677	1016	698	430
30	L	4158	2520	1732	1066
	M	3289	1994	1370	843
	Q	2358	1429	982	604
	H	1782	1080	742	457
31	L	4417	2677	1840	1132
	M	3486	2113	1452	894
	Q	2473	1499	1030	634
	H	1897	1150	790	486
32	L	4686	2840	1952	1201
	M	3693	2238	1538	947
	Q	2670	1618	1112	684
	H	2022	1226	842	518
33	L	4965	3009	2068	1273
	M	3909	2369	1628	1002
	Q	2805	1700	1168	719
	H	2157	1307	898	553
34	L	5253	3183	2188	1347
	M	4134	2506	1722	1060
	Q	2949	1787	1228	756
	H	2301	1394	958	590
35	L	5529	3351	2303	1417
	M	4343	2632	1809	1113
	Q	3081	1867	1283	790
	H	2361	1431	983	605

36	L	5836	3537	2431	1496
	M	4588	2780	1911	1176
	Q	3244	1966	1351	832
	H	2524	1530	1051	647
37	L	6153	3729	2563	1577
	M	4775	2894	1989	1224
	Q	3417	2071	1423	876
	H	2625	1591	1093	673
38	L	6479	3927	2699	1661
	M	5039	3054	2099	1292
	Q	3599	2181	1499	923
	H	2735	1658	1139	701
39	L	6743	4087	2809	1729
	M	5313	3220	2213	1362
	Q	3791	2298	1579	972
	H	2927	1774	1219	750
40	L	7089	4296	2953	1817
	M	5596	3391	2331	1435
	Q	3993	2420	1663	1024
	H	3057	1852	1273	784

J4L Micro QRCode for the Java[TM] platform

Introduction

The package J4L-Micro QRCode contains the classes you need to create Micro QRCode barcodes within your jsp or java applications.

The Micro QR Code format (also specified in this International Standard), is a variant of QR Code with a reduced number of overhead modules and a restricted range of sizes. A single finder pattern, is located at the upper left corner of the symbol as illustrated in the following figure.

Micro QR Code symbols have 4 version (version 1 to 4). The sizes of the barcodes are:

- version M1: 11 x 11 modules.
- version M2: 13 x 13 modules.
- version M3: 15 x 15 modules.
- version M4: 17 x 17 modules.

The maximum barcode capacity of Micro QR Code the largest barcode symbol size, (Version 4 error correction level L):

- numeric data (digits 0-9): 35 characters
- aphanumeric data (digits 0 - 9 , upper case letters A -Z and nine other characters: space, $ % * + - . /): 21 characters
- Byte data ((bytes 0-255): 15 characters
- Kanji data (hexadecimal values 8140 -9FFC and E040 - EBBF): 9 characters

Supports 4 error correction levels:

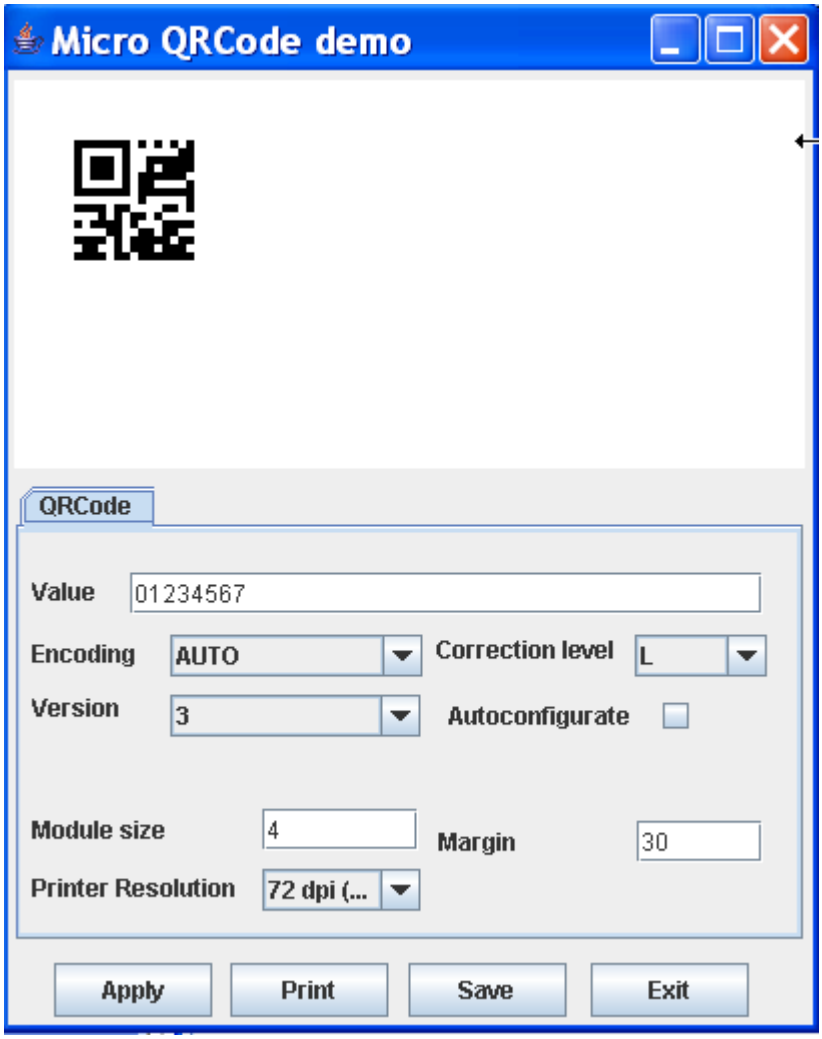
- L (7% of the symbol codewords).
- M (15% of the symbol codewords).
- Q (25% of the symbol codewords). Only in version M4.

Installation and requirements

In order to install the software you just need to unzip the ZIP file in an empty directory (your must unzip with pathname so that the subdirectories are properly created.

Examples

In order to run the sample application you must execute **runDEMO.bat**. The sample application requires JDK 1.3 or later.



In the sample application you can set all properties of the Micro QRCode symbology.

You can execute the following commands:

- Apply: repaint the symbol using the new properties.
- Print: send image to printer at the selected resolution.
- Save: save the symbol in jpg format. We recommend you to use gif or png formats,however the sample application uses JPG because it does not require you to download any additional package. In order to create gif or png files you must download an external encoder (see javadoc).

- Exit: terminate application.

The Servlet

Micro QRCode comes with a standard servlet (*com.java4less.microqrcode.QRCodeServlet*) you can use to create Micro QRCode images. The parameters for the servlet can be sent using GET or POST commands. The parameters are:

- **AUTOCONFIGURATE:** if true (value Y) the preferredVersion can be ignored if the data does not fit in the selected version.
- **BAR_COLOR**: color of the bars. You can also use the RGB numeric value of a color as parameter. For example "0xFF0000" is red (RGB format), "00x00FF00" is green and so on.
- **BACK_COLOR**: color of the background.
- **DEBUG**: use ON to activate verbose output.
- **ERROR_CORRECTION**: valid values are L, M or Q.
- **ENCODING**: encoding mode (default is AUTO). Valid values are:
  - AUTO: Automatic selecting of the encoding method.
  - ALPHA: encode alpahnumeric characters only (upper case letter plus 9 additional characters).
  - NUMERIC: encode alpahnumeric digits only.
  - BYTE: use this mode to encode binary data.
  - KANJI: encodes Kanji characters only.
- **PROCESSTILDE**: if true (value Y) the tilde character (~) will be processed like this:
  - ~~: will be replaced with ~
  - ~dxxx: will be replaced by the character whose ascii code is xxx. For example ~d065 will be replaced with A.
- **DATA**: string to be encoded.
- **MARGIN**: left and top margin in pixels (default is 30).
- **MODULE_SIZE**: number of pixels which make a module (square) in the barcode (default is 4).
- **WIDTH**: size of the image. Default is 400.
- **HEIGHT**: size of the image. Default is 200.
- **FORMAT:** output format: PNG, GIF or JPEG (default):
  - The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
  - The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar
- **CONFIGURATION**: preferred format (version). Another version will be automatically selected if AutoConfigurate=true and the amount of data and the selected error correction level does not fit in the preferred version. Valid values are 1 to 4.

For example, if you want to test the servlet using Tomcat 4.1 you must do the following:

1. copy the qrcode.jar to */tomcat/webapps/examples/Web-inf/lib*
2. Start the server and test the servlet entering the following address and parameters in your browser:

   http://localhost:8080/examples/servlet/MicroQRCodeServlet?DATA=ABC&ENCODING=NORMAL

In order to property print a barcode created with the servlet you must use the <IMG> tag. In this way you can embed the image in your HTML page. Note that you will need to use the attibutes height and width in order to achieve the correct size of the barcode on the printed page.

This is a simple HTML page that contains a qrcode symbol:

```
<HTML>
<HEAD>

        <TITLE>Servlet Example META http-equiv=Content-Type
        content="text/html; charset=windows-1252">

</HEAD>
<BODY bgColor=#ffffff>

        This is your Barcode:
        <IMG height=100 width=100
        src="http://localhost:8080/examples/servlet/MicroQRCodeServlet?DATA=TEST" >

</BODY>
</HTML>
```

The Applet

You must add the following code in your html pages to use the applet:

```
<APPLET
CODEBASE = "./"
CODE =
"com.java4less.microqrcode.QRCodeApplet.class"
ARCHIVE = "microqrcode.jar"
NAME = "TestApplet"
WIDTH = 300
HEIGHT = 250
HSPACE = 0
VSPACE = 0
ALIGN = middle
>

<PARAM NAME = "DATA" VALUE = "123411">

</APPLET>
```

Note that the microqrcode.jar file must be located in the directory referenced by the CODEBASE parameter.

The available paramete are the same as those used by the servlet (see description in the previous section):

- **AUTOCONFIGURATE**
- **BAR_COLOR**
- **BACK_COLOR**
- **ERROR_CORRECTION**
- **ENCODING**
- **PROCESSTILDE**
- **DATA**
- **MARGIN**
- **MODULE_SIZE**
- **CONFIGURATION**

Annex A

Table — Data capacity for Micro QRCode versions

Version	Error Correction Level	Numeric	Alphanumeric	Byte	Kanji
1		5	-	-	-
2	L M	10 8	6 5	-	-
3	L M	23 18	14 11	9 7	6 4
4	L M Q	35 30 21	21 18 13	15 13 9	9 8 5

- Alphanumeric mode is not available in Version M1 Micro QR Code symbols.
- Byte mode is not available in Version M1 or M2 Micro QR Code symbols.
- Kanji mode is not available in version M1 or M2 Micro QR Code symbols.

J4L Maxicode for the Java[TM] Platform

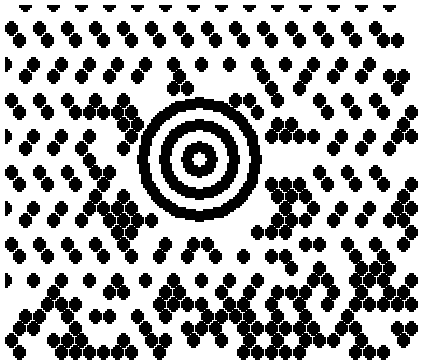
Introduction

The package RMaxicode contains the classes you need to create Maxicode barcodes within your java applications.

Maxicode is a two-dimensional (2D) and fixed-size matrix symbology which is made up of offset rows of hexagonal modules arranged around a unique finder pattern. The size of Maxicode is 1.11 x 1.054 inches and it can contain up to 93 data characters of information (a total of 144 including error correction codewords).

The symbol was created by United Parcel Service **for fast scanning** and sorting of packages.RMaxicode supports the following features:

- modes 2,3,4,5 and 6.
- structured append
- full character set.



Symbol structure

MaxiCode symbols are divided into a primary and a secondary message, each of which contains data and error correction codewords. The primary message is made of 10 codewords and 10 error correction codewords. The primary message is used in mode 2 and 3 to encode the Postal Code, the Service class and the country.

Error correction

MaxiCode symbols have error correction codewords, based on Reed-Solomon error correction algorithms, which can be used to detect errors and correct erroneous symbols. There are two error levels; standard and enhanced (mode 5).

Maxicode modes

Modes 2 and 3 are designed for use in the transport industry. They encode the destination address and the class of service as defined by the carrier. Mode 2 uses a 9 digit numeric postal code and mode 3 uses a 6 character alphanumeric postal code. The Service class and the country are both a 3 digit number in both modes.

Mode 4 can encode up to 93 characters or 138 digits. Mode 5 can only encode up to 77 characters but it provides more error correction capabilities (enhanced mode than mode 4.

Mode 6 indicates that the symbol encodes a message used to program the reader system (scanner).

Structured append

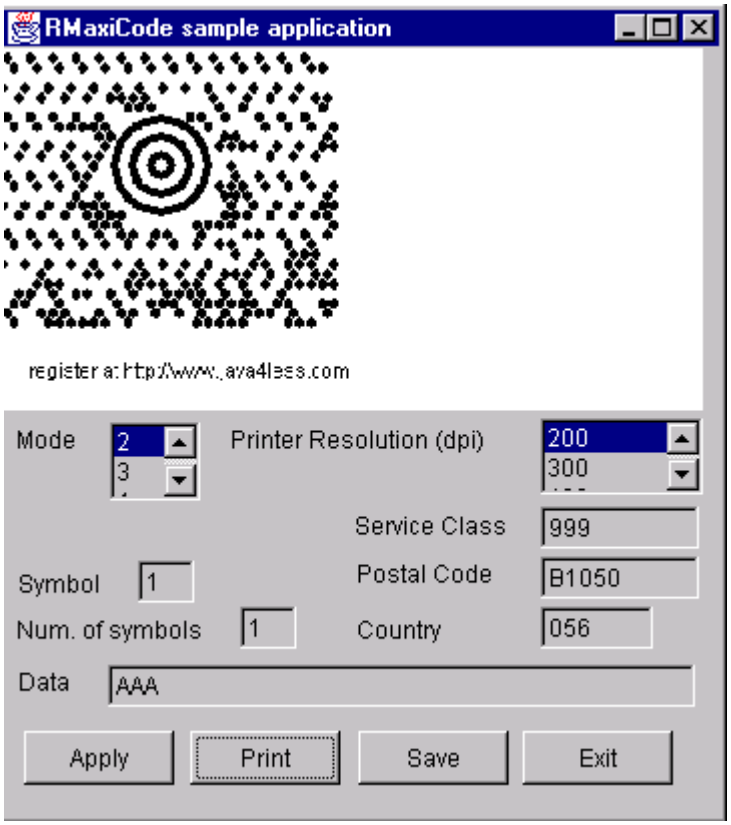
Up to eight MaxiCode symbols may be appended in a structured format. This allows files of data to be represented in up to 8 MaxiCode symbols.

Sample application

In order to run the sample application you must execute **example.bat**. You need to use **JDK 1.3** ot later. The reason is the following:

> Maxicode must be printed at a minimum resolution of 200 dpi. The sample application uses the *PageAttributes* class of java 1.3 in order to set the resolution of the printer and print the barcode.

Note: the sample application needs java 1.3 or later



In the sample application you can set following properties:

- Mode to be used
- Printer resolution (dots per inch).
- Service class, Country and Postal code (for modes 2 and 3).
- Data to be encoded.
- Number of symbols in the secuence (if you use structured append)
- Position of the symbol within the secuence.

and execute the following command:

- Apply: repaint the symbol using the new properties.
- Print: print using the given resolution.
- Save: save the symbol in GIF format using the given resolution.

Note: Tilde Processing: You can use the format **~ddd** if you want to specify the ascii code of the character to be encoded. For example, if you enter the following text in the Data field:

~066AA

you will actually be encoding

BAA

because ~066 will be replaced with the character ascii character 66, which is B. You must use ~~ in order to encode the ~ character. If you want to deactivate this behaviour you must call the method setProcessTilde(false).

The Servlet

RMaxicode comes with a standard servlet (RMaxicodeServlet) you can use to create maxicode images. The paramters for the servlet can be sent using GET or POST commands. The parameters are:

- MODE: values 2 to 6.
- POSTALCODE.
- COUNTRY.
- SERVICE.
- DATA. Data to be encoded.
- ROTATION. rotation angle, 0, 90, 180 or 270.
- NUMBER. Number of symbols in the secuence (if you use structured append).
- POSITION. Position of the symbol within the secuence.

For example, if you want to test the servlet using JSWDK 1.0 you must do the following:

1. copy the RMaxicode classes to */jswdk-1.0.1/examples/Web-inf/servlets*
2. Add the following line to the file */jswdk-1.0.1/examples/Web-inf/servlets.properties* :

   RMaxicodeServlet.code=com.java4less.rmaxicode.RMaxicodeServlet

3. Start the server an test the servlet entering the following address and parameters in your browser:

   http://localhost:8080/examples/servlet/RMaxicodeServlet?ZIPCODE=B1050&SERVICE=999&COUNTRY=056&DATA=TEST

Important Note:

In order to property print a maxicode barcode created with the servlet you must use the <IMG> tag. In this way you can embed the image in your HTML page. Note that you will need to use the attibutes height and width in order to achieve the correct size of the maxicode on the printed page.

This is a simple HTML page that contains a maxicode barcode:

```
<HTML>
<HEAD>

        <TITLE>Servlet Example META http-equiv=Content-Type
        content="text/html; charset=windows-1252">
```

```
</HEAD>
<BODY bgColor=#ffffff>

        This is your Maxicode:
        <IMG height=100 width=100
        src="http://localhost:8080/examples/servlet/RMaxicodeServlet?ZIPCOD
        E=B1050&SERVICE=999&COUNTRY=056&DATA=TEST" >

</BODY>
</HTML>
```

References

You can find more information about maxicode at:

- Maxicode
- AIM International, Inc.
- ISO 16023

RSS and EAN-UCC Composite Barcodes (GS1 Databar) for the Java[TM] Platform

Introduction

This package can be used in java applications, applets and web server for creating RSS and EAN-UCC composite barcodes. These are also known as GS1 Databar barcodes. The supported symbologies are:

- RSS. The EAN.UCC Reduced Space Symbology (RSS) family contains three linear symbologies:
  - **RSS-14** encodes a 14-digit EAN.UCC item identification. This symbology has 4 formats , regular, truncated, stacked and omni-directional stacked.
  - **RSS Limited** encodes a 14-digit EAN.UCC item identification with Indicator digits of zero or one in a linear symbol for use on small items.
  - **RSS Expanded** encodes EAN.UCC item identification plus supplementary AI element strings such as weight and “best before” date in a linear symbol. This symbology has 2 formats, regular and stacked.
- EAN-UCC Composite: these symbols are intended for encoding identification numbers and data supplementary to the identification. Each barcode consists of a linear component and a multi-row 2D Composite Component. The 2D Composite Component is printed above the linear component. The linear component can be any EAN-UCC or RSS linear symbol. The 2D component can be only of the following:
  - **CC-A**, encodes up to up to 56 digits.
  - **CC-B**, encodes up to up to 338 digits.
  - **CC-C**, up to 2 361 digits.

Supported combinations of linear and multi-row symbols are:

- UPC-A and EAN-13 with a CC-A or CC-B 2D component (4 columns).
- EAN8 with a CC-A or CC-B 2D component (3 columns).
- UPCE with a CC-A or CC-B 2D component (2 columns).
- EAN128 with a CC-A or CC-B 2D component (4 columns).
- EAN128 with a CC-C component.
- RSS-14 with a CC-A or CC-B 2D component (4 columns).
- RSS-14 Stacked with a CC-A or CC-B 2D component (2 columns).
- RSS-14 Stacked Omni-directional with a CC-A or CC-B 2D component (2 columns).
- RSS Limited with a CC-A or CC-B 2D component (3 columns).

- RSS Expanded with a CC-A or CC-B 2D component (4 columns).
- RSS Expanded Stacked with a CC-A or CC-B 2D component (4 columns).

Java API

This section explains how to use the classes in your java application. You will find a complete description of all classes and methods in the **Javadoc** files.

In order to create a barcode you must create an instance of one of the following classes:

- ***com.java4less.rss.BarCode***: use this class to create EAN8, EAN13, EAN128, UPCE or UPCA symbols. Use the *setSymbology()* method to select the symbology to use.
- ***com.java4less.rss.RSS***: use this class to create RSS14, RSS stacked, RSS truncated or RSS stacked omni-directional symbols. Use the *setRSSFormat()* method to select the symbology to use.
- ***com.java4less.rss.RSSLimited***: use this class to create RSS Limited symbols.
- ***com.java4less.rss.RSSExpanded*** : use this class to create RSS Expanded or Expanded Stacked symbols. Use the *setRSSFormat()* method to select the stacked or non-stacked format.

The following example creates a EAN128 barcode and exports it to a jpg file.

```
BarCode r=new BarCode();
r.setSymbology(BarCode.EAN128);
r.setCode("0193812345678901");
r.setHumanReadableCode("(01)93812345678901");
r.setSize(300,300);
new ImageEncoder(r,"JPEG","c:\\barcode.jpg");
```

If you want to create a composity symbols you must set the value for the 2D component using the *setSecondaryCode()* method. For example:

```
RSS r=new RSS();
r.setCode("0341234567890"); // Note: do not pass the 01
application identifier to RSS or RSSLimited classes.
r.setSecondaryCode("17010200"); // this forces the 2D component
to be created.
r.setRSSFormat(r.FORMAT_STACKED);
r.setSize(300,300);
new ImageEncoder(r,"JPEG","c:\\barcode.jpg");
```

Note that you should not pass the 01 application identifier nor the checksum character in the *setCode()* method of the RSS or RSSLimited classes. You must however include all application identifiers for EAN128 and RSS Expanded symbols.

All linear symbols (except EAN128) can be associated with a CC-A or CC-B 2D component. The software will automatically select the correct component depending on the amount of data you need to encode.

EAN128 can be painted with a CC-A/B component or a CC-C component. In this case you must select the desired symbology using the *setEAN128WithCCC()* method. If you select CC-C you must also set the number of columns using *setCCCColumns()* (the default value is 4):

```
BarCode r=new BarCode();
r.setSymbology(r.EAN128);
r.setCode("0193812345678901");
r.setHumanReadableCode("(01)93812345678901");
// create a composite barcode
r.setEAN128WithCCC(true);
r.setSecondaryCode("10ABCD123456#4103898765432108");
r.setSize(300,300);
new ImageEncoder(r,"JPEG","c:\\barcode.jpg");
```

How to create a gif, png or jpg file.

You can also export the barcode to a gif,png or a jpeg file. In order to do this you must use the following code:

```
import com.java4less.rbarcode.*;

bc=new BarCode();
bc.setSize(400,200); // important, set size
....
new ImageCodeEncoder(bc,"GIF","file.gif");
new ImageCodeEncoder(bc,"PNG","file.png");
new ImageCodeEncoder(bc,"JPEG","file.jpg");
```

note that:

- The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
- The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar
- In order to use the JPEG encoder you must have the package com.sun.image.codec.jpeg installed. This feature is only available in JDK 1.2 or later.

How to create the barcode in a java.awt.Image object

The following code illustrates how you can paint a barcode in a java.awt.Image object:

```
bc=new BarCode();
bc.setSize(400,200); // important, set size

// create image
java.awt.image.BufferedImage image = new
```

```
java.awt.image.BufferedImage(
bc.getSize().width,bc.getSize().height,java.awt.image.BufferedI
mage.TYPE_BYTE_INDEXED );

// get graphic context of image
java.awt.Graphics imgGraphics = image.createGraphics();

// paint barcode in graphics context of image
bc.paint(imgGraphics );
```

How to use RBarcode in a web site

You have two possibilities:

- **Use the applet** to create barcodes in the browser. In this case you must create HTML dinamically. The HTML page will contain the applet and the parameters.
- **Use a servlet** to create a gif or jpeg image in the server and send the to the browser.

com.java4less.rss.BCApplet

- You can use BCApplet to display barcodes in your HTML pages.

The available parameters for the applet are:

- CODE_TYPE: Use this parameter to select the symbology. The value must be one of the following: EAN8, EAN13, EAN128 , UPCE, UPCA, RSS14, RSSLIMITED or RSSEXPANDED.
- RSS_FORMAT: Use this parameter to select the format of RSS14 or RSS expanded symbols. The accepted values are: FORMAT_REGULAR, FORMAT_TRUNCATED, FORMAT_STACKED, FORMAT_STACKED_OMNIDIRECTIONAL, FORMAT_EXPANDED, FORMAT_EXPANDED_STACKED.
- EAN128_CCC (Y or N): selects the 2D component to be painted together with EAN128 symbols. Use Y to select CC-C and N to select CC-A/B.
- CCC_COLUMNS: number of columns of the CC-C component (only used if EAN128_CCC=Y).
- SECONDARY_CODE: value to be encoded in the 2D Component (use # as FNC1 character).
- PRIMARY_CODE: value to be encoded in the linear component (use # as FNC1 character).
- READABLE_CODE: Value to be painted (as human readable text) below the linear code. If empty the package will calculate it (except for EAN128).
- SUPPLEMENT_SEPARATION: separation in pixels between the symbol and the supplement.
- SUPPLEMENT_HEIGHT: height of the bars of the supplement.
- SUPPLEMENT: Type of EAN supplement 2 or 5
- ROTATE: Indicates how the barcode should be painted (vertica, horizontal ...). Valid values are 0 (normal), 90 (vertical),180 (inverted) and 270 (inverted vertical).
- X: width of the bars in pixels
- LEFT_MARGIN: left margin in pixels.
- TOP_MARGIN: top margin in pixels.
- BAR_COLOR: color of the bars.
- PROCESS_TILDE. If Y, the ~ character will be processed as follows:
    - ~dNNN stands for the character whose ascii code is NNN. For example ~d065 stands for 'A'.
    - ~~ stands for ~
- FONT_COLOR: color of the font used to display the code.

- BACK_COLOR: back color of the barcode.
- GUARDBARS: indicates if guardbars will be height than other bars. Only for EAN and UPC.
- UPCE_SYSTEM: encoding system to be used for UPCE, valid values are 0 and 1 (default).
- TEXT_FONT: font used to display the human readable code. Set this value to NULL if you want to suppress the text.
- H: Indicates how to calculate the height of the bars. A value of 0.5 means that the bars should be half the length of the symbol.
- BAR_HEIGHT: instead of using the H paramneter you can use this parameter to specify the height of the bars of the linear component (in pixels):
- BAR_HEIGHT2D: height in pixels of the rows of the 2D Component.

Some parameters of the applet have a special format:

- Colors: valid values are: RED,BLUE,GREEN,BLACK,GRAY,LIGHTGRAY,WHITE,DARKGRAY,YELLOW,ORANGE,CYAN and MAGENTA. You can also use the RGB numerical value of a color as parameter (e.g. 0x00FF00 if green).
- Fonts have the format <font name>|<style>|<size>. Style can be PLAIN, ITALIC or BOLD. Example: "Arial|BOLD|12"

Example of how to use the applet:

```
<APPLET
CODEBASE = "./"
CODE = "com.java4less.rss.BCApplet.class"
ARCHIVE1 = "rrss.jar"
NAME = "TestApplet"
WIDTH = 300
HEIGHT = 250
HSPACE = 0
VSPACE = 0
ALIGN = middle
><PARAM NAME = "PRIMARY_CODE" VALUE = "0361234567890">
<PARAM NAME = "CODE_TYPE" VALUE = "RSS14">
<PARAM NAME = "LEFT_MARGIN" VALUE = "10">
<PARAM NAME = "TOP_MARGIN" VALUE = "10">
<PARAM NAME = "TEXT_FONT" VALUE = "ARIAL|BOLD|11">
</APPLET>
```

You can provide the parameters in the Applet PARAM tag or you can also do it from **Javascript**. For example, the following code set a new value for the barcode:

```
TestApplet.setParameter('PRIMARY_CODE','new value');
TestApplet.refresh();
```

There in an example of the use of javascript in the AppletExample.html file.

The servlet

RBarcodeServlet will allow you to use RBarcode as Servlet without any Java programming. The servlet has the advantage that the java classes must not be downloaded to the client's browser. This means your barcode will be displayed faster. It has however the disadvantage that you need to have a web server able to execute servlets.

In the case of servlets, the barcodes are created in the server and the output in PNG, GIF or JPEG format is sent to the browser. This also means that you can use RBarcode to in browsers that do not support Java.

You can very easily use RBarCodeServlet. The parameters are the same as those for the applet. You can send the parameters to the Servlet using the POST or GET methods. Furthermore there are some additional parameters:

- WIDTH: width in pixels of the output image. (default is 500).
- HEIGHT: height in pixels of the output image. (default is 500).
- FORMAT: format of output image, "gif", "png" or "jpeg". (default is JPEG).

Note that

- RBarCodeServlet needs jdk 1.2 or newer.
- The GIF encoder must previously be downloaded and included in your classpath: http://www.acme.com/resources/classes/Acme.tar.gz
- The PNG encoder must previously be downloaded and included in your classpath: http://209.51.137.74/pngencoder-1.0.jar

Example of installation on Tomcat 5.0:

1. create directory *rss* in the *webapps* directory (i.e. *tomcatDirectory\webapps\rss*)
2. copy the *rrss.jar* to *tomcatDirectory\webapps\rss\WEB-INF\lib*
3. copy the *web.xml* file from any other project to *tomcatDirectory\webapps\rss\WEB-INF\web.xml*. Add the following lines to the file:

```
<servlet>
<servlet-name>RSSServlet</servlet-name>
<servlet-class>com.java4less.rss.RBarCodeServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>RSSServlet</servlet-name>
<url-pattern>/servlet/RSSServlet</url-pattern>
</servlet-mapping>
```

4. Start the server (tomcatDirectory\bin\startup.bat) an test the servlet entering the following address and parameters in your browser:

```
http://localhost:8080/rss/servlet/RSSServlet?DEBUG=ON&PRIMARY_CODE=03
61234567890&LEFT_MARGIN=20&CODE_TYPE=RSS14&WIDTH=300&HEIGHT=300
```

```
http://localhost:8080/rss/servlet/RSSServlet?DEBUG=ON&PRIMARY_CODE=03
61234567890&LEFT_MARGIN=20&CODE_TYPE=RSS14&SECONDARY_CODE=1234
```

GS1 Standards

GS1-128

GS1-128 barcode was previously known as EAN128 and can be found in our Barcide 1D library.

GS1 128 barcodes have the following structure:

- They start with a special character called FNC1 (added automatically by the library)
- After data the data to be encoded is a list of pair:
  - Application identifier
  - Field Data (which can be have a variable or constant length). The variable length fields which have a separator at the end.
- And a final check character added by the library

In the example below the human readable text displays the application identifiers enclosed within the ( ) signs.

The following Java code can be used to generate the above GS1 128 barcode:

```
BarCode bc = new BarCode();
bc.setSize(400,250);
bc.code="01250123456789041707122 5";
bc.I = 1;
bc.resolution=1;
bc.X=1;
bc.barType=BarCode.EAN128;
bc.checkCharacter=true;
```

If you have variable length application identifier, you have to add a space after the data field so that the library knows which the data field finishes.

GS1 Datamatrix

GS1 DataMatrix is a specific use of Data Matrix ECC 200 used GS1 data elements like item identification, expiration date, etc.

In oder to create a GS1 compliant barcode the data to be encoded must meet this criteria:

- the data to be encoded must start with a leading FNC1 character
- The GS1 Application Identifiers (AI) are used for all data
- only ascii characters are used.

How to achieve this using our Datamatrix component?

- use ~1 to encode the FNC1 in the first position
- use ~1 as field separator for variable length AI elements.
- use ascii encoding

For example, if you want to encode these two elements

1. Application identifier 10 (Batch or Lot number) and value ABCD1234
2. Application identifier 410 (Ship To - Deliver To) and value 9501101020917

you would use in our Java component this code:

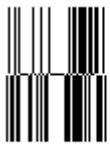
```
RDataMatrix rdatamatrix = new RDataMatrix();
rdatamatrix.setSize(300, 300);
rdatamatrix.code = "~110ABCD1234~14109501101020917";
rdatamatrix.barType = RDataMatrix.DATAMATRIX;
rdatamatrix.dotPixels = 3;
rdatamatrix.encoding = RDataMatrix.E_ASCII;
rdatamatrix.processTilde=true;
```

note the value to be encoded is ~110ABCD1234~14109501101020917. The value starts with ~1 and it contains another ~1 as field separator because the AI 10 is a variable length filed.

GS1 Databar

GS1 is a set of barcode symbologies that was formerly named **RSS**.

The following table shows the old and new names of the symbologies:

former RSS name	current GS1 Name
RSS-14	GS1 Databar Omnidirectional  (01)03612345678904 GS1 Databar Omnidirectional (RSS-14) encodes a 14-digit EAN.UCC item identification. It can be can be scanned omnidirectionally by point-of-sale scanners.
RSS-14 truncated	GS1 Databar Truncated  (01)03412345678900 This is the regular GS1 Databar, except its height is reduced to a 13X. It is small for small items.
RSS-14 Stacked	GS1 Databar Stacked  (01)03412345678900 The stacked version is a GS1 Databar truncated two-row barcode.
RSS-14 Stacked Omnidirectional	GS1 Databar Stacked Omnidirectional  (01)20012345678909 GS1 Databar Stacked Omnidirectional (RSS-14 Stacked) is a variation of the regular symbology that is stacked in two rows. This is used when the

	regular GS1 Databar barcode is too wide.
RSS Limited	GS1 Databar Limited  The limited symbology encodes a 14-digit EAN.UCC item identification with Indicator digits of zero or one .
RSS Expanded	GS1 Databar Expanded  The expanded version encodes the EAN.UCC item identification and supplementary application identifier elements. It can encode 74 numeric or 41 alphabetic characters.
RSS Expanded Stacked	GS1 Databar Expanded Stacked  Stacked version of the expanded barcode.

The following Java code shows how to create a GS1 Databar stacked omnidirectional:

```
RSS bc=new RSS();
bc.setRSSFormat(RSS.FORMAT_STACKED_OMNIDIRECTIONAL);
bc.setCode("2001234567890");
```

J4L Barcodes plugin for Apache FOP (generation of PDF files)

Introduction

Apache FOP is an implementation of the XSL formatting objects (XSL-FO) which takes an input XML file and creates PDF files. It can also create other output formats but the most popular is PDF.

The J4L Barcode plugin for Apache FOP allows you to add barcodes to the Apache FOP documents. This allows you to create PDF files which contain the following types of barcodes:

- Barcode 1D (see http://www.java4less.com/barcodes/barcodes.php)
- PDF417 (option of Macro PDF417 available)
- Datamatrix
- QRCode
- AztecCode
- Maxicode

The plugin not a standalone product but isincluded in the regular J4L Barcoding components license.

Installation

In order to use the plugin in your FOP documents you need to add the following jars to your classpath:

- rbarcode-fop.jar
- rbarcode.jar
- qrcode.jar, raztec.jar or rmaxicode.jar (if you need to create any of these barcodes)

Sample application

We deliver a simple application which creates PDF files using as input a FOP file:

- In the *examples* subdirectory you will find a set of **.fo* files used as template for creating the PDF files
- The file *ExampleFO2PDF.java* is the sample application which will take the **.fo* files and will create the PDF files.

How to run the sample application:

1. Create a Java Project with your favorite IDE tool (Eclipse, Netbeans ...)
2. Add *ExampleFO2PDF.java* to your project
3. Add *rbarcode-fop.jar, lib/rbarcode.jar* and *lib/qrcode.jar* to the project´s classpath
4. Download Apache FOP (we tested 0.93 and 0.95) and add all fop jar files to your classpath
5. Set the working directory of your project to be the *examples* subdiretory (so that the **.fo* files can be read by the application)
6. Run the application, it will create a set of pdf files in the working directory

How to add barcode to a FOP / PDF file

If you are not familiar with FOP, please read this tutorial first. If you are familiar with FOP, you know FOP files are XML files which describe the content of the target PDF file. In order to add a barcode to a FOP file you use these element:

```
<fo:instream-foreign-object >

<j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop"
mode="inline">

<barcode1d>

<value>This is a code128</value>
<type>CODE128</type>
<set>A</set>
<X>1</X>
<margin>30</margin>
<text>Readable text</text>
<barHeight>40</barHeight>
<leftMargin>40</leftMargin>
<topMargin>50</topMargin>
<barColor>BLUE</barColor>
<backColor>YELLOW</backColor>
<fontColor>RED</fontColor>

</barcode1d>

</j4lbarcode>

</fo:instream-foreign-object>
```

The blue elements are always the same for any type of barcode, the green elements will change depending on the kind of barcode you want to create (barcode 1D, pdf417 , datamatrix or qrcode). The example above creates a Barcode 1D.

For Datamatrix you would use:

```
<fo:instream-foreign-object >
<j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop" mode="inline">

<datamatrix>

    <code>This is a Datamatrix</code>
    <moduleSize>2</moduleSize>
    <processTilde>false</processTilde>
    <margin>30</margin>
    <encoding>AUTO</encoding>
    <format>C24X24</format>

</datamatrix>

</j4lbarcode>
</fo:instream-foreign-object>
```

For PDF417:

```
    <fo:instream-foreign-object >
    <j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop"
    mode="inline">

    <pdf417>

    <code>This is a PDF417</code>
    <rows>0</rows>
    <maxRows>30</maxRows>
    <cols>5</cols>
    <ecLevel>3</ecLevel>
    <compaction>TEXT</compaction>
    <X>1</X>
    <H>10</H>
    <margin>30</margin>

    </pdf417>

    </j4lbarcode>
    </fo:instream-foreign-object>
```

for QRCode:

```
<fo:instream-foreign-object >
<j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop" mode="inline">
```

```
<qrcode>

    <code>This is a QRCode</code>
    <moduleSize>2</moduleSize>
    <processTilde>false</processTilde>
    <margin>30</margin>
    <ecLevel>H</ecLevel>
    <encoding>AUTO</encoding>
    <configuration>1</configuration>

</qrcode>

</j4lbarcode>
</fo:instream-foreign-object>
```

for Maxicode:

```
<fo:instream-foreign-object >
<j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop" mode="inline">

    <maxicode>
    <code>1Z12345679~d029UPSN~d029123X56~d029187
    ~d029~d0291/1~d02910~d029N~d029~d029COLOGNE~
    d029~d030</code>
    <country>276</country>
    <serviceclass>066</serviceclass>
    <zipcode>51147</zipcode>
    <mode>3</mode>
    <processTilde>true</processTilde>
    </maxicode>

</j4lbarcode>
</fo:instream-foreign-object>
```

and for AztecCode:

```
<fo:instream-foreign-object >
<j4lbarcode xmlns="http://java4less.com/j4lbarcode/fop" mode="inline">

    <azteccode>
    <!-- <CODEBINARY>BASE64 value</CODEBINARY>  --
    >
    <code>1234567890</code>
```

```
        <moduleSize>2</moduleSize>
        <processTilde>false</processTilde>
        <configuration>18</configuration>
        <margin>30</margin>
        <encoding>NORMAL</encoding>
        </azteccode>

    </j4lbarcode>
    </fo:instream-foreign-object>
```

This section provides now a short description of the meaning of the elements that define the barcode:

- **Barcode 1D**:
  - **type**: type of barcode to create. Valid values are: BAR39, BAR38EXT, CODE128, CODE11, CODABAR,CODE93, CODE93EXT, MSI, IND25, MAT25, INTERLEAVED25, EAN13, EAN8, EAN128, UPCA, UPCE or POSTNET.
  - **value**: value to be encoded
  - **set**: value A, B or C (for type CODE128 only)
  - **processTilde**.See J4L Barcodes documentation
  - **text**: value to be displayed below the barcode.
  - **barHeight**: height of bars in pixels
  - **X**: width of narrow bars in pixels
  - **N**: multiplier for wide bars. A value of 2, means that wide bars will be 2 times the width of narrow bars. The default vaue is 2
  - **barCodor**. Color of bars. Valid values are RED, BLACK, BLUE, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, PINK, WHITE or YELLOW. You can also use the RGB value.
  - **fontColor**: color of the text.
  - **backColor**: background color.
  - **font:** font of the text, for example "ARIAL|BOLD|10", the format is <family|style|size>
  - **leftMargin**: margin in pixels
  - **topMargin**: margin in pixels

- **QRCode**
  - **code**: value to be encoded.
  - **moduleSize**: size (pixels) of the modules (dots) of the matrix.
  - **margin**: margin in pixels.
  - **ecLevel**: can be H, L, M or Q.
  - **encoding**: can be ALPHA, BYTE, NUMERIC, KANJI or AUTO.
  - **configuration**: the configuration is the size of the qrcode. Valid values are 1 to 40, set 1 for automatic configuration selection.

- **Datamatrix**
  - **code**: value to be encoded.
  - **processTilde**: See QRCode documentation in evaluation version.
  - **moduleSize**: size (pixels) of the modules (dots) of the matrix.

- **margin**: margin in pixels
- **encoding**: The default is ASCII. Posible values are ASCII, C40, TEXT, BASE256 and AUTO.
- **format**: If empty the format will be selected automatically, if not you can specify the format (e.g. C24X24).

- **PDF417**

  - **code**: value to be encoded
  - **rows**: number of rows
  - **maxRows**: maximum number of rows
  - **cols**: number of columns
  - **ecLevel**: error correction level
  - **compaction**: can be NUMERIC, TEXT or BINARY
  - **X**: width of narrows bars in pixels
  - **H**: height of bars
  - **margin**: margin in pixels

- **Maxicode**

  - **code:** data to be encoded. In modes 2 and 3 if the postal code is empty it will be extracted from the begining of this string (see maxicode specifications). Use ~d029 as GS separator, ~d030 as EOT character, ~d062 as RS character and [) for code begin.
  - **mode:** maxicode mode. Values range from 2 to 6.
  - **resolution:** resolution of the printer in dpi, Default is 200. Other valid vaues are 300,400,500 and 600.
  - **processTilde.**
  - **country:** country code (modes 2 and 3.
  - **serviceClass:** service class (modes 2 and 3). In mode 2 it can be an alphanumeric of length 5. In mode 3 can be a numeric of length 9.
  - **zipCode:** postal code (modes 2 and 3).
  - **L:** Set length of length symbol. The default is 25.5 millimeters.

- **Azteccode**

  - **code:** value to be encoded
  - **CODEBINARY:** binary value to be encoded
  - **moduleSize:** size (pixels) of the modules (dots) of the matrix.
  - **processTilde:** see AztecCode documentation
  - **margin:** margin in pixels.
  - **ecLevel:** default is 23 (23%)
  - **encoding:** can be NORMAL and BINARY
  - **type:** 0-any, 1-compact, 2-full

Jasper Report and iReport Integration

You can use our barcode component in Jasper Reports in a very easy way.

If you are using Jasper Report without iReport

1. Use the <*image*> tag to add an image to your report
2. In the *imageExpression* part you enter the class *net.sf.jasperreports.engine.JRRenderable*.
3. In the content of the expression you enter a reference to our components.

This is a very simple report which includes a barcode:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport
PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport name="Simple_Report">
<detail>
<band height="500">
<staticText>
<reportElement x="20" y="0" width="200" height="20"/>
<text><![CDATA[This is my first Barcode:]]></text>
</staticText>

<image hyperlinkType="None">
<reportElement x="20" y="20" width="100" height="50"/>
<imageExpression
class="net.sf.jasperreports.engine.JRRenderable">
<![CDATA[ new com.java4less.rbarcode.jr.J4LBarcodeRenderer(
(new
com.java4less.rbarcode.BarCodeFacade()).createBarcodeImage
("CODE128","1234567890","A","false","1234567890",0,20,1,2,nu
ll,null,null,null,30,30,null) )]]>
</imageExpression>
</image>

</band>
</detail>
</jasperReport>
```

the Java code to run this report would be (we assume the previous report has been copied to the file Report.jrxml):

```
import java.util.HashMap;
import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
```

```
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;

public class MyReport {

public static void main(String[] args)
{
JasperReport jasperReport;
JasperPrint jasperPrint;
try
{

        jasperReport =
        JasperCompileManager.compileReport("Report.jrxml
        ");
        jasperPrint =
        JasperFillManager.fillReport(jasperReport, new
        HashMap(), new JREmptyDataSource());
        JasperExportManager.exportReportToPdfFile(jasper
        Print, "report.pdf");


}
catch (JRException e)
{
e.printStackTrace();
}
}
}
```

You will find some examples in the jasperreports subdirectory of the delivery.

The expression used for the other barcode types are:

- PDF417

  new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new **com.java4less.rbarcode.BarCode2DFacade**()).createBarcodeImage("This is a PDF417",null,0,0,3,1,"TEXT",1,4,20,null))

- Datamatrix

  new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new **com.java4less.rdatamatrix.RDataMatrixFacade**()).createBarcodeImage("This is a Datamatrix",null,false,3,30,"AUTO",null,null))

- QRCode

  new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new **com.java4less.qrcode.QRCodeFacade**()).createBarcodeImage("This is a QRCode",null,3,false,20,"H","AUTO",1,null))

- AztecCode

  new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new

com.java4less.raztec.AztecCodeFacade()).createBarcodeImage("data",null,3,false,20,23,"BINARY",2,1,null))

- Maxicode

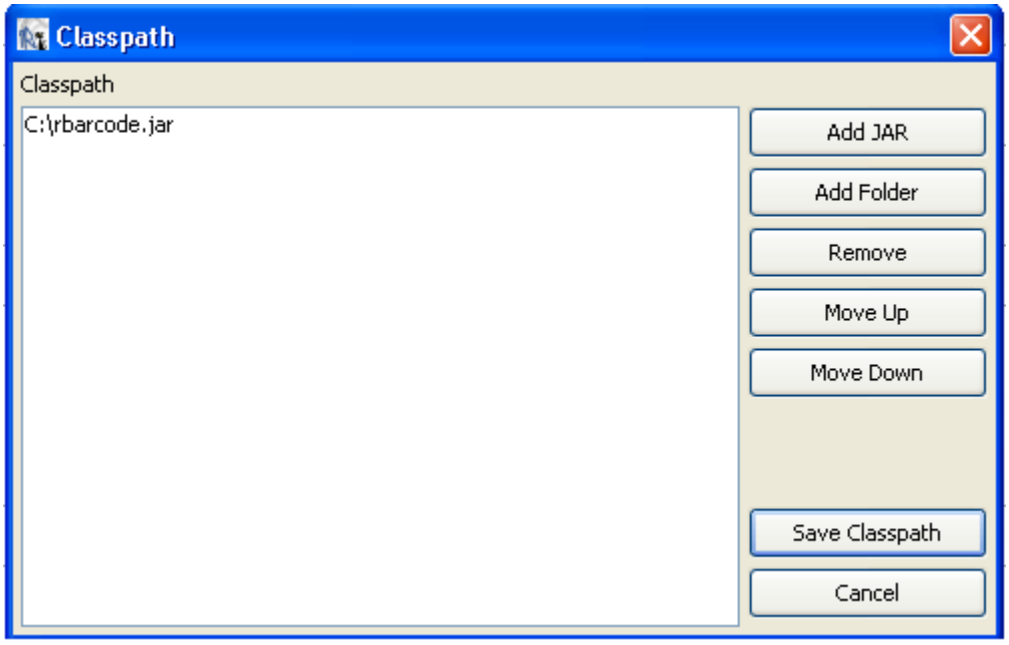
  new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new **com.java4less.rmaxicode.RMaxiCodeFacade**()).createBarcodeImage("AAA",2,200,false,"056","B1050","999",25.5,null))

Please check the *JavaDoc* subdirectory in the evaluation version. You will find there the documentation of the *Facade classes and the description of the paramters.

If you are using iReport

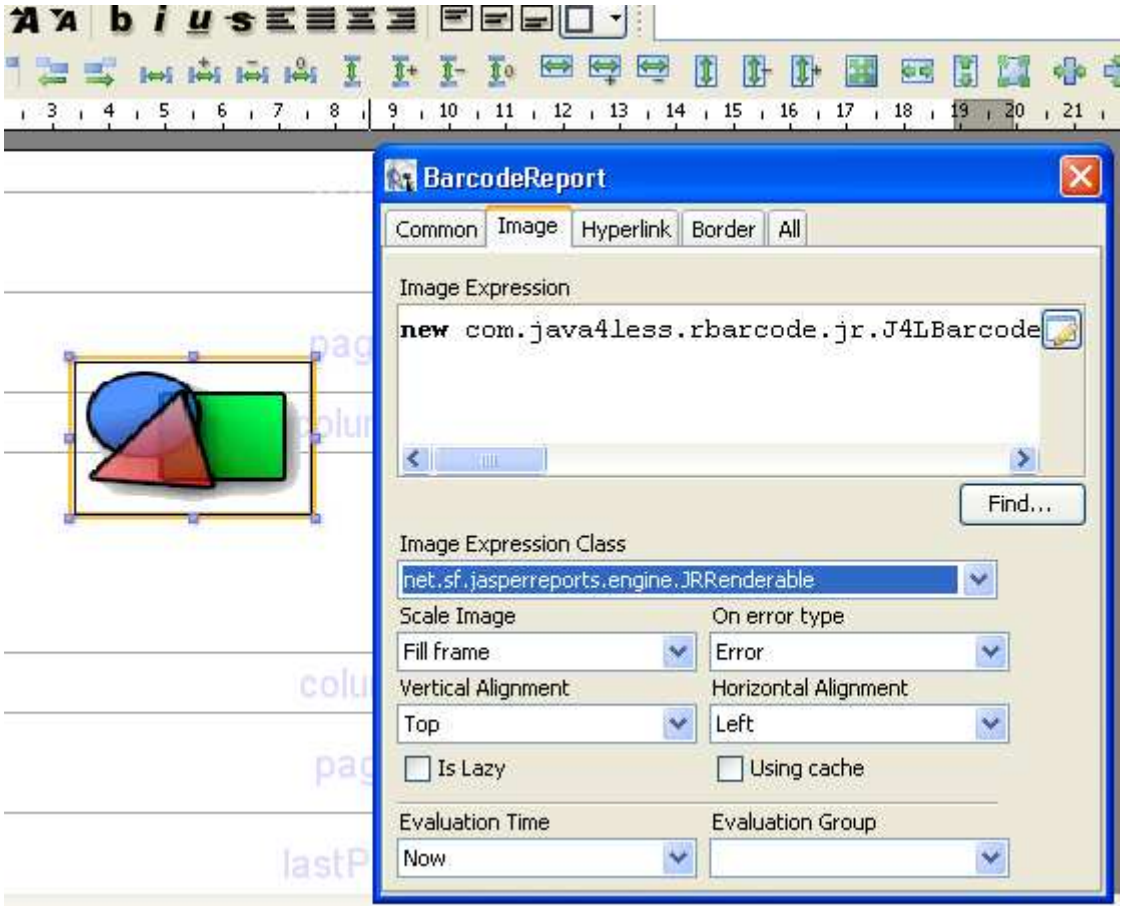
In this case you have to:

1. Add rbarcode.jar and/or qrcode.jar to the reports classpath by selection *main menu, options , classpath*



2. Add an image to your report
3. In the image properties select *Image Expression Class= net.sf.jasperreports.engine.JRRenderable*
4. In the Image expression field enter one of the expression as explained in the previous section, for example:

   new com.java4less.rbarcode.jr.J4LBarcodeRenderer((new com.java4less.rbarcode.BarCodeFacade()).createBarcodeImage("CODE128", "1234567890","A","false","1234567890",0,20,1,2,null,null,null,null,30,30,null))


BarcodeReport
Common
Image
Hyperlink
Border
All
Image Expression
new com.java4less.rbarcode.jr.J4LBarcode
Find...
Image Expression Class
net.sf.jasperreports.engine.JRRenderable
Scale Image
Fill frame
On error type
Error
Vertical Alignment
Top
Horizontal Alignment
Left
Is Lazy
Using cache
Evaluation Time
Now
Evaluation Group

Jasper Reports custom components

The barcoding components can also be used as a jasper reports custom component.

There are components for linear barcodes, PDF417, Datamatrix, QRCode, Aztec code and Maxicode. Other can be added on demand.

In order to use the components you need to add to your class path these 2 files:

- The jar file belonging to the barcode java library (for example qrcode.jar or rbarcode.jar)
- And the Jasper Report component library j4ljrbarcode.jar

In the *jaspercomponents* subdirectory of our download version you will find an example (jrxml file) for each one of the components.

In the same directory you will also find the xml schema files (*.xsd) where you can also find documentation about the attributes of the barcode.

The example belows shows how a EAN128 barcode could be defined:

```
<componentElement>
<reportElement x="0" y="70" width="500" height="300" uuid="d60723de-a9ab-
4d33-84a8-8ac759234088"/>
      <bc:Barcode1D xmlns:bc="http://barcode.com/jasperreports/barcode1d"
xsi:schemaLocation="http://barcode.com/jasperreports/barcode1d
http://barcode.com/jasperreports/barcode1d/barcode1d.xsd"

                        evaluationTime="Report"
                        foreColor="#0000FF"
                        backColor="#00FFFF"
                        fontColor="#000000"
                        textFont="Arial;BOLD;12"
                        processTilde="true"
                        textOnTop="false"
                        rotation="0"
                        barHeight="20"
                        barWidth="1"
                        checksum="true"
                        code128Charset="A"
                        symbology="13"
                        bcMargin="30"     >

      <bc:codeExpression><![CDATA["Data"]]></bc:codeExpression>
      <bc:codeTextExpression><![CDATA["Data"]]></bc:codeTextExpression>

      </bc:Barcode1D>
</componentElement>
```

The following Java code shows how to run one of the reports:

```
import java.util.HashMap;
import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;

public class MainApp
{


     public static void main(String[] args) throws JRException
     {

         JasperReport jasperReport;
         JasperPrint jasperPrint;
         try
         {
          jasperReport =
          JasperCompileManager.compileReport("qrcode.jrxml");

           jasperPrint =
          JasperFillManager.fillReport(jasperReport, new
          HashMap(), new JREmptyDataSource());

          JasperExportManager.exportReportToPdfFile(jasperPrint
          , "qrcode.pdf");

         }
         catch (JRException e)
         {
           e.printStackTrace();
         }
     }



}
```

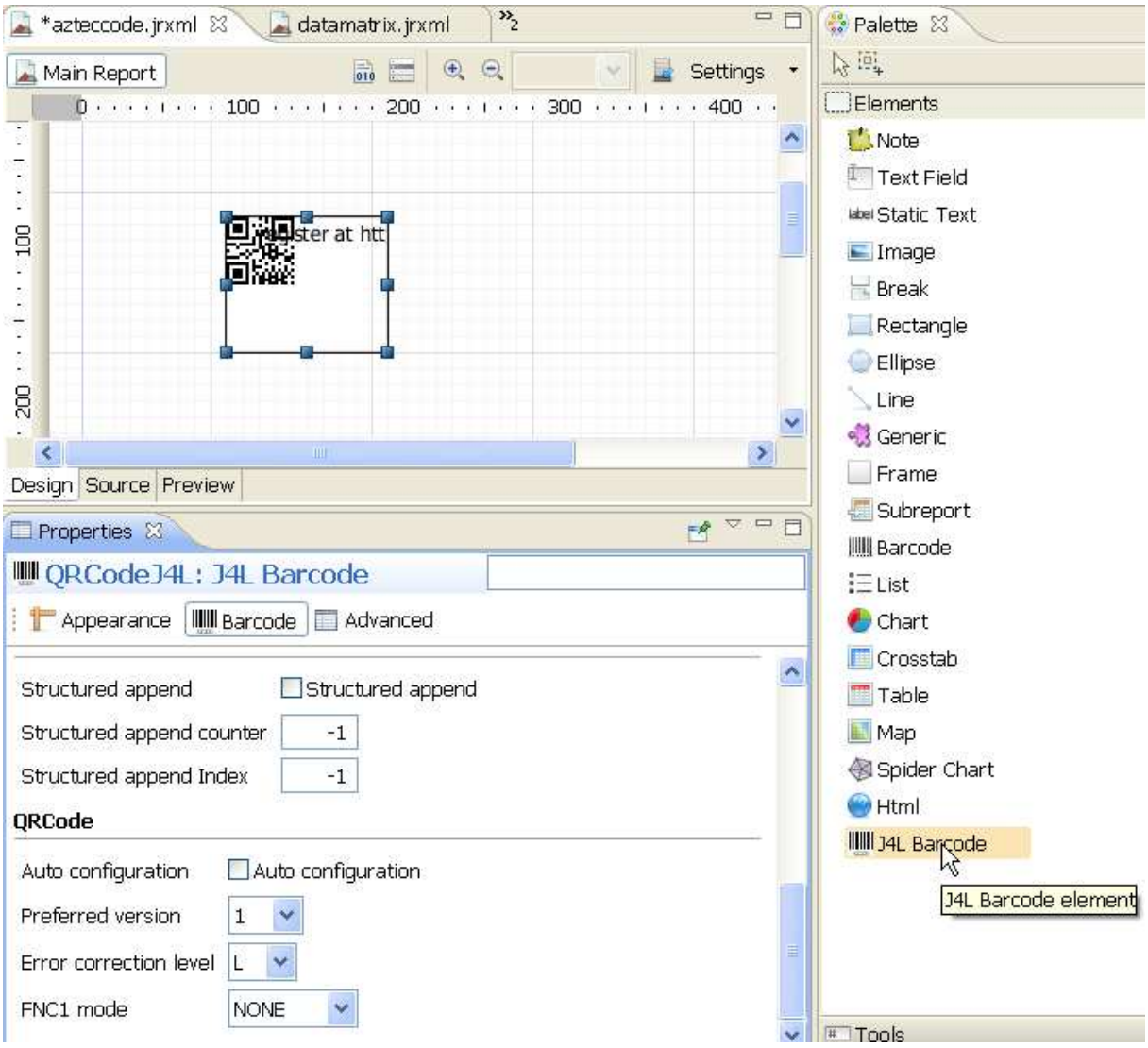
Jaspersoft Studio

Starting with Jasperstudio 5.5 you can add the barcode components without manual coding. Download our demo version copy the file

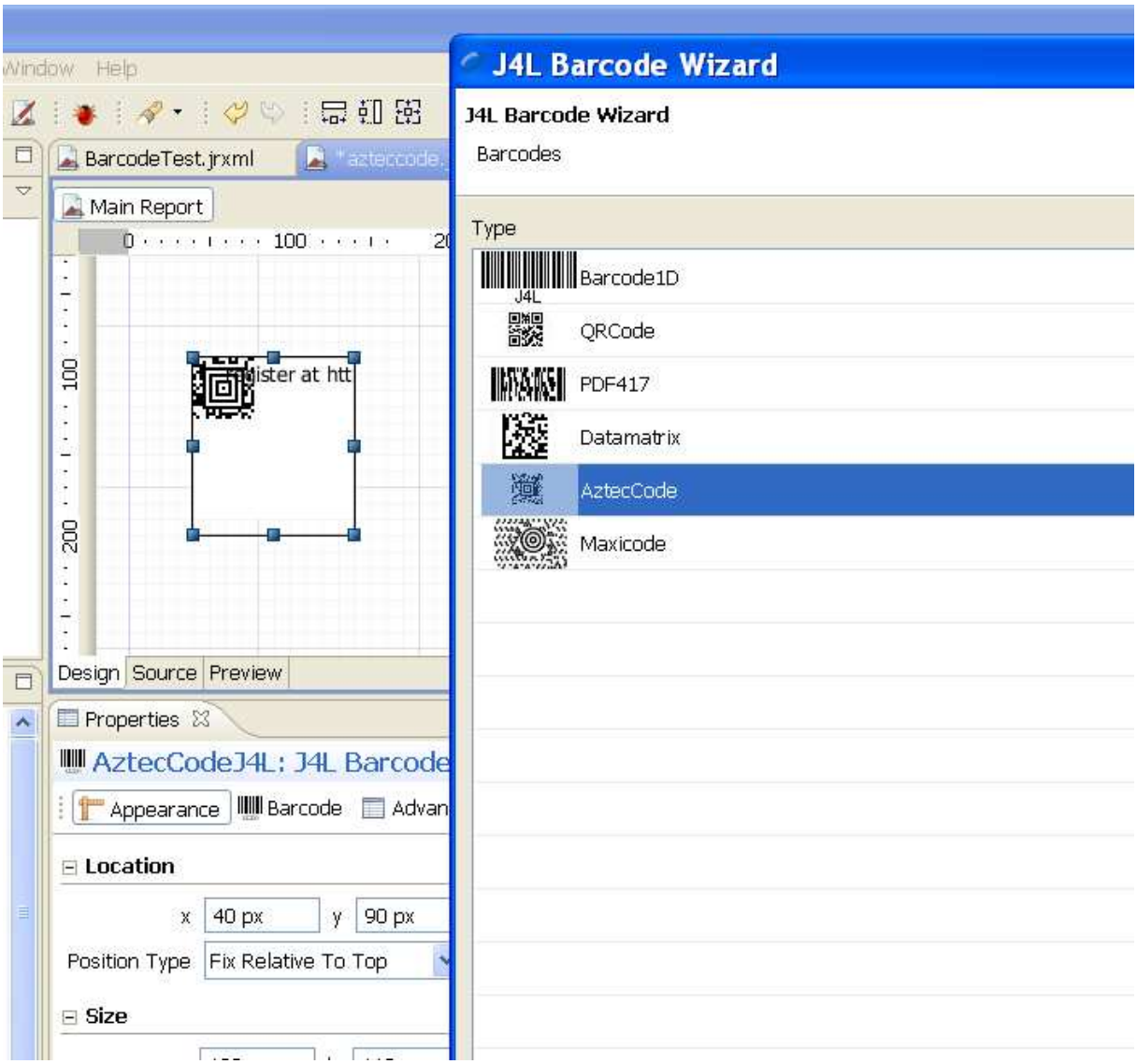
com.java4less.jaspercomponents_5..jar*

to the Jasperstudio "*dropins*" subdirectory.

After restarting the studio you will see the barcode elements in the Palette



After selecting the element in the palette the barcode wizard will display the available barcodes.



J4L Barcoding Web Services

Introduction

The J4L Barcode web services are **Java EE** web services that create barcode images. The following symbologies are currently supported:

- Barcode 1D (see http://www.java4less.com/barcodes/barcodes.php)
- PDF417 (option of Macro PDF417 available)
- Datamatrix
- QRCode

The web services are not a standalone product but are included in the regular J4L Barcoding components.

The web services receive an input XML document containing the parameters of the barcode to be created and returns an XML document that contains the barcode image. The transmission of the XML documents between the client and the web service is implemented using SOAP/HTTP protocol.

An example of a barcode creation request document is:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<ns2:createBarcode xmlns:ns2="http://ws.datamatrix.java4less.com/">

        <code>This is a Datamatrix</code>
        <processTilde>false</processTilde>
        <moduleSize>4</moduleSize>
        <margin>10</margin>
        <encoding>ASCII</encoding>
        <format>C16X16</format>

</ns2:createBarcode>
</S:Body>
</S:Envelope>
```

an example of the barcode image (JPEG format and base64 encoded) returned by the web service is:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<dlwmin:createBarcodeResponse
xmlns:dlwmin="http://ws.datamatrix.java4less.com/">


<BarcodeImage>iVBORw0KGgoAAAANSUhEUgAAAGQAAABkCAIAAAD/gAIDAAAB
xElEQVR42u3a0ZLCIAxAUf7/p/VNR1tSQpIS0suD47CtSw8xpMX2og23BgFYYIEFFlh
gQQAWWMmx2m877e8dv2O/4KDD+pzzQCx1ZMlY7dB6/fIpn9feX+Uwl/vlye69n4ms3
kUKg5BxB4m1KNrJ+BuPD9bcjLWxdpzYkVMuh6SK3NOR3BdZlxd/HNBEZBXBuoyv0
6+AKidOLERaKf/VUJXgBTI53dyP9ROPqshSDVS7VI/8r5HjteeG1Fnj+aIY1jdO3SOrHt
```

bk7Q5Y6jrLeJHGr3no54MFFljPwooGtSfpmx7+gQUWWBmewU8nS0u/9hhLAwsssGp
iRSTaVZOR7qkDWGCBFYplKSa9HvhFF9VggQXWfliOe3AbFaJggQVWBSyvZBx9o2
6ZPLDAAmtvrOiNhojNVHtSBwsssHbF8rpJ9r0Ah63TVashWGCBlaoojU72cT+0BQsss
CpgRSTjbIUrWGCB9Swsr+LTkrxTbFiABRZYZbBWbXbYFwqwwAKrGpZXwek1AakT
PFhggbX891nRDwu9CtR0uztggQXWbXXWqg3X6Bt1sMACqw5W1QYWWGDtjUUD
CyywwAILLBpYYIGVqr0BNV+ZhbGDz+4AAAAASUVORK5CYII=</BarcodeImage>
</dlwmin:createBarcodeResponse>
</soapenv:Body>
</soapenv:Envelope>

Since the communication between client and server (web service) is pure HTTP/SOAP based, the web service is language independent and can be consumed from any web service client application, like a .NET application or Javascript, which do not need to understand Java.

Requirements

The web services components run on a **Java EE server** (Java 1.5 or later). The following servers have been tested:

- Sun Application Server 9.1 (Glassfish v2)
- IBM Websphere Application Server Community Edition v2.1
- JBoss 4.2.3GA

Since components used only standard functions, they should run on any certified Java EE server. Other servers like SAP Netweaver 7.1 and Oracle Application Server 11g will be tested also. Let us know if you use one of these.

Deployment

Deployment is the installation of the web service in a Java EE server. The web service is just a file with **.war** extension that needs to be installed in the server. There is one of such war files for each barcode symbology we support.

The deployment process depends on the server you use but here you have some simple directions:

- Sun Application Server (Glassfish). Open the adminitration console in the URL http://yourserver:4848 , login, select web *applications* and select *deploy*.
- IBM Websphere: Start the administration console from the windows IBM Websphere menu , login, select *deploy new* from the applications main menu.
- JBoss: just drop the war in the *jboss/server/default/deploy* directory and it will be installed automatically.

Once the web service has been deployed you can see the web service description (WSDL file) opening once of the following URL, we assume your server is running on the localhost , port 8080, if that is not the case you need to modify the URL:

- Barcode 1D: http://localhost:8080/j4lrbarcode/BarcodeWS?wsdl
- QRCode: http://localhost:8080/j4lqrcode/QRCodeWS?wsdl
- PDF417: http://localhost:8080/j4lpdf417/PDF417WS?wsdl

- Datamatrix: http://localhost:8080/j4ldatamatrix/DatamatrixWS?wsdl

Sample client applications

We deliver a set of standalone Java applications which will call the web service and display the result in a Swing JFrame component. These applications are located in the */clients* subdirectory.

In order to run the clients you would need to create some proxy classes. These proxy classes must be created with a tool named *wsimport*. This tool reads the WSDL files (see the URL in the previous section) and creates some Java classes. The generated classes take care of the XML and SOAP details so for your application, calling a web service is just calling a method in a Java class. For example:

```
URL url=new URL("http://localhost:8080/j4lqrcode/QRCodeWS?wsdl");
// Note this is the dynamic approach, we provide the URL of the web
service at runtime.
Service service2=Service.create(url, new
QName("http://ws.qrcode.java4less.com/","QRCodeWS"));
QRCodeWS port = service2.getPort(QRCodeWS.class);
java.awt.Image result = port.createBarcode("This is a QRCode",
4,false,10,"H","ALPHA",1);
```

In this example the QRCodeWS class is one of the proxy classes generated by wsimport.

We deliver the generated proxy classes in the *clientProxies.jar* file, so you do not have to worry about creating them with the wsimport tool (but you can if you want). The web service infrastructure is provided by the Java Metro open source product. You need to download this before running the clients (we used Metro 1.2) and make sure the *metro/lib/webservices*.jar* files are included in your classpath.

Of course you can also use other web service stack like Apache Axis2 (instead of Metro).

Parameters of the request

If you open the URL of the WSDL deployed in your server you will see the file starts with:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns="http://schemas.x. .....
<types>
<xsd:schema>
<xsd:import namespace="http://ws.rbarcode.java4less.com/"
```

```
schemaLocation="http://localhost:8080/j4lrbarcode/BarcodeWS?xsd=../../WEB-INF/wsdl/BarcodeWS.xsd" />
</xsd:schema>
</types>
```

the *schemaLocation* attribute gives you the URL where you can see the schema of the XML document used as request to create a barcode. The XML schema describes the format of the document used in the web service request, this is similar to the method signature in a Java class which describes the parameters used in a method.

In this example the schema's URL would be:

http://localhost:8080/j4lrbarcode/BarcodeWS?xsd=../../WEB-INF/wsdl/BarcodeWS.xsd

if you open that XSD file you will see this complexType:

```
<xs:complexType name="CreateBarcode">
<xs:sequence>

    <xs:element minOccurs="0" name="type"
    type="xs:string" />
    <xs:element minOccurs="0" name="value"
    type="xs:string" />
    <xs:element minOccurs="0" name="set"
    type="xs:string" />
    <xs:element minOccurs="0"
    name="processTilde" type="xs:string" />
    <xs:element minOccurs="0" name="text"
    type="xs:string" />
    <xs:element name="rotate" type="xs:int" />
    <xs:element name="barHeight"
    type="xs:int" />
    <xs:element name="X" type="xs:int" />
    <xs:element name="N" type="xs:int" />
    <xs:element minOccurs="0"
    name="barColor" type="xs:string" />
    <xs:element minOccurs="0"
    name="fontColor" type="xs:string" />
    <xs:element minOccurs="0"
    name="backColor" type="xs:string" />
    <xs:element minOccurs="0" name="font"
    type="xs:string" />
    <xs:element name="leftMargin"
    type="xs:int" />
    <xs:element name="topMargin"
    type="xs:int" />

</xs:sequence>
</xs:complexType>
```

these are the parameters of the Barcode 1D creation request.

This section provides now a short description of the meaning of the parameters without using the XML schema format:

- **Barcode 1D**:
  - **type**: type of barcode to create. Valid values are: BAR39, BAR38EXT, CODE128, CODE11, CODABAR,CODE93, CODE93EXT, MSI, IND25, MAT25, INTERLEAVED25, EAN13, EAN8, EAN128, UPCA, UPCE or POSTNET.
  - **value**: value to be encoded
  - **set**: value A, B or C (for type CODE128 only)
  - **processTilde**.See Barcode 1D documentation.
  - **text**: value to be displayed below the barcode.
  - **barHeight**: height of bars in pixels
  - **X**: width of narrow bars in pixels
  - **N**: multiplier for wide bars. A value of 2, means that wide bars will be 2 times the width of narrow bars. The default vaue is 2
  - **barColor**. Color of bars. Valid values are RED, BLACK, BLUE, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, PINK, WHITE or YELLOW. You can also use the RGB value.
  - **fontColor**: color of the text.
  - **backColor**: background color.
  - **font:** font of the text, for example "ARIAL|BOLD|10", the format is <family|style|size>
  - **leftMargin**: margin in pixels
  - **topMargin**: margin in pixels

- **QRCode**

```
<xs:complexType name="createBarcode">
<xs:sequence>
<xs:element minOccurs="0" name="code" type="xs:string" />
<xs:element name="moduleSize" type="xs:int" />
<xs:element name="processTilde" type="xs:boolean" />
<xs:element name="margin" type="xs:int" />
<xs:element minOccurs="0" name="ecLevel" type="xs:string" />
<xs:element minOccurs="0" name="encoding" type="xs:string" />
<xs:element name="configuration" type="xs:int" />
</xs:sequence>
</xs:complexType>
```

  - **code**: value to be encoded.
  - **processTilde**: See QRCode documentation in evaluation version.
  - **moduleSize**: size (pixels) of the modules (dots) of the matrix.
  - **margin**: margin in pixels.
  - **ecLevel**: can be H, L, M or Q.
  - **encoding**: can be ALPHA, BYTE, NUMERIC, KANJI or AUTO.
  - **configuration**: the configuration is the size of the qrcode. Valid values are 1 to 40, set 1 for automatic configuration selection.

- **Datamatrix**

```
<xs:complexType name="createBarcode">
<xs:sequence>
<xs:element minOccurs="0" name="code" type="xs:string" />
<xs:element name="processTilde" type="xs:boolean" />
<xs:element name="moduleSize" type="xs:int" />
<xs:element name="margin" type="xs:int" />
<xs:element minOccurs="0" name="encoding" type="xs:string" />
<xs:element minOccurs="0" name="format" type="xs:string" />
</xs:sequence>
</xs:complexType>
```

  - **code**: value to be encoded.
  - **processTilde**: See J4L Barcodes documentation
  - **moduleSize**: size (pixels) of the modules (dots) of the matrix.
  - **margin**: margin in pixels
  - **encoding**: The default is ASCII. Posible values are ASCII, C40, TEXT, BASE256 and AUTO.
  - **format**: If empty the format will be selected automatically, if not you can specify the format (e.g. C24X24).

- **PDF417**

```
<xs:complexType name="createBarcode">
<xs:sequence>
<xs:element minOccurs="0" name="code" type="xs:string" />
<xs:element name="rows" type="xs:int" />
<xs:element name="maxRows" type="xs:int" />
<xs:element name="cols" type="xs:int" />
<xs:element name="ecLevel" type="xs:int" />
<xs:element minOccurs="0" name="compaction" type="xs:string"
/>
<xs:element name="X" type="xs:int" />
<xs:element name="H" type="xs:int" />
<xs:element name="margin" type="xs:int" />
</xs:sequence>
</xs:complexType>
```

  - **code**: value to be encoded
  - **rows**: number of rows
  - **maxRows**: maximum number of rows
  - **cols**: number of columns
  - **ecLevel**: error correction level
  - **compaction**: can be NUMERIC, TEXT or BINARY
  - **X**: width of narrows bars in pixels
  - **H**: height of bars
  - **margin**: margin in pixels

Getting help

Send an email to ***java4lessATconfluencia.net*** if you have questions