# MACRO PDF417 2D BARCODE ENCODER AND DECODER

**A PROJECT REPORT**

***Submitted by***

| | |
|---|---|
| **M.V NEEMA** | **11307104061** |
| **M.PAVITHRA** | **11307104066** |
| **R.SANGEETHA** | **11307104084** |

***In partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

IN

**COMPUTER SCIENCE AND ENGINEERING**

**APRIL 2011**

## ANNA UNIVERSITY: CHENNAI 600 025

## BONAFIDE CERTIFICATE

Certified that this project report **“MACRO PDF417 2D ENCODER AND DECODER”** is the bonafide work of “**M.VNeema (11307104061), M.Pavithra(11307104066), R.Sangeetha (11307104084)”** who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **DR. K.L.SHUNMUGANATHAN M.E.,Ph.D** | **R. JAGADEESH KANNAN B.E.,M.S.,** |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| Professor | Senior Lecturer |
| Department of Computer Science and Engineering, | Department of Computer Science and Engineering, |
| R.M.K Engineering College, | R.M.K Engineering College, |
| R.S.M Nagar, Kavaraipettai. | R.S.M Nagar, Kavaraipettai. |
| Gummidipoondi Taluk, | Gummidipoondi Taluk, |
| Thiruvallur - 601206. | Thiruvallur – 601206. |

Submitted for the project VIVA-VOCE held on _____________

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

## ABSTRACT

Barcode symbology is a technique which is used to encode  text into barcode which is a sequence of lines and spaces.There are various barcode symbologies existing like code49,code 93,pdf417 etc.,PDF417 is an effective barcode symbology which uses reed Solomon for error correction levels. But the challenges faced in PDF417 is it cant be used to encode large files. Thus we propose a system Macro PDF417  which  is an implementation of PDF417 capable of encoding very large amounts of data into multiple PDF417 barcodes. These multiple barcodes are then scanned by a MacroPDF enabled scanner, which reassembles them into one string of data. Each MacroPDF barcode segment shares a file ID with the other segments, which make up the entire barcode sequence. If the file ID is not the same, the scanner will assume that each of the segments belong to a different barcode sequence.Macro PDF417 has high data capacity and offers Reed-Solomon error correction, which makes it highly secure. It can also be read with a laser scanner in many applications.

## TABLE OF CONTENTS

| CHAPTER NO: | TITLE | PAGENO. |
|---|---|---|
| | **ABSTRACT** | iii |
| | **LIST OF FIGURES** | |
| | **LIST OF TABLES** | |
| | **LIST OF ABBREVIATIONS** | |
| **1** | **INTRODUCTION** | |
| | **1.1** SYSTEM OVERVIEW | |
| | **1.2** OBJECTIVE | |
| | **1.3** LITERATURE SURVEY | |
| **2** | **SYSTEM REQUIREMENTS&SPECIFICATION** | |
| | **2.1** PROBLEM STATEMENT | |
| | **2.1.1** Existing System | |
| | **2.1.2** Proposed System | |
| | **2.1.3** System Requirement | |
| | 2.2 OVERVIEW OF THE APPROACH | |
| **3** | **DESIGN & IMPLEMENTATION** | |
| | **3.1** DESIGN DETAILS | |
| | **3.2** DATA FLOW DIAGRAMS | |
| | **3.3** IMPLEMENTATION DETAILS | |
| **4** | **RESULTS & DISCUSSION** | |
| | **4.1** TEST CASES | |
| | **4.2** DISCUSSION | |

**5** **CONCULSION AND FUTURE WORK**

**6** **APPENDICES**

**6.1** APPENDIX 1

**6.2** APPENDIX 2

**6.3** APPENDIX 3

**6.4** APPENDIX4

**7** **REFERENCES**

# LIST OF TABLES

| **TABLE NO.** | **TITLE .** | **PAGENO.** |
|---|---|---|
| | Advantages and disadvantages of Barcode Symbologies | |
| | Comparison of different Barcodes | |
| | Error Correction Capacity | |

## LIST OF FIGURES

| FIGURE NO. | TITLE | PAGE NO |
|---|---|---|
| | Existing System | |
| | Overall Architecture | |
| | Symbol Structure | |
| | Macro PDF417 Example | |
| | Level 0 DFD | |
| | Level 1 DFD | |
| | Level 2  DFD | |

## List of Symbols, Abbreviations and Nomenclature

| SYMBOL | MEANING |
|---|---|
| PDF | Portable Data File. |
| ECC | Error Correction Capacity |

# 1. INTRODUCTION

## 1.1 SYSTEM OVERVIEW

Macro PDF417, or Macro Portable Data File 417, has the same features as PDF417 standard. Macro PDF417 provides a standard mechanism for creating a distributed representation of data files too large to be represented by set of single PDF417 symbols. Macro PDF417 symbols differ from ordinary PDF417 symbols in that they contain additional control information in a zone called Macro PDF417 Control Block. Using Macro PDF417, large data files are split into several data segments and encoded into individual symbols. The Macro PDF417 decoder device uses the Control Block's information to reconstruct the file correctly, independent of symbol scanning order.

## 1.2 OBJECTIVE

The main objective of this project is to encode and  decode large files in to sequence of barcode using java technology and make it available in open source.. When a file is given as input it should generate the appropriate barcode sequence and the receiver must be able to decode it and get back the file content. So that these barcodes can be used for transfering files which are highly confidential and it can be used in e governance etc.

## 1.3 **LITERATURE SURVEY**

### Linear Bar Codes

Literally hundreds of linear symbologies are in use today. However, only a few are widely adopted and appropriate for use in the Life Sciences. Listed below are some of the more common symbologies employed.

**UPC/EAN** was developed by the Uniform Code Council (UCC) for use in the retail grocery industry. It has several good features, including a mandatory check digit and good density. *Advantage:* The UPC code has good data security and immunity to ink spread when "similar-edge" algorithm is used. *Disadvantage:* It has a limited number of characters (numeric characters only) and is used for point-of-sale applications.

**Code 39** was the first alphanumeric symbology and is still widely used today, Code 39 is also recommended by the NCCLS for specific applications.[2] *Advantage:* Code 39 offers good security and an alphanumeric character set. *Disadvantage:* Since it is not as dense as Code 128, Code 39 is not recommended for applications requiring long data identifiers.

**Code 128** is a very reliable symbology offering three different character sets. *Advantage:* Its large character set combined with high density and flexibility has earned it the recommendation of NCCLS[3] and ISBT,[4] among others. Code 128's mandatory check character maximizes data integrity. *Disadvantage:* Since Code 128 is a modular symbology, users must be aware that problems can arise when the narrow bar width approaches the resolution of the bar code generation system.

**Codabar** was originally developed for the blood bank industry, but the industry has now started to use Code 128. While Codabar is still used in a number of applications, it offers no advantages and several disadvantages. The code's most significant drawbacks are very low security and a limited character set.

**Interleaved 2 of 5** is widely used in shipping applications. *Advantage:* It is the highest density linear bar code for numeric only strings of fewer than 10 characters. *Disadvantage:* Interleaved 2 of 5 has very low security. The symbology uses two bar patterns for start and stop instead of full characters, which can lead to truncation errors.

**Code 93** uses the same character set as Code 39. Because Code 93 is a modular symbology, it produces a much more compact bar code than ratio-based Code 39. *Advantage:* Code 93 always uses two check digits to ensure data security. It is also the densest linear symbology for random alphanumeric strings. *Disadvantage:* The symbology is not a standard symbology read by most scanners, and its varying dimensions can be difficult to print accurately.

**RSS** is a new family of linear symbologies recently introduced by the UCC for small, space-constrained applications. *Advantage:* The RSS family is the most compact linear bar code symbology offered today and has limited orientation requirements. *Disadvantage:* RSS uses multiple bar code widths. As with other modular symbologies, users must be aware that difficulties can arise when the narrow bar width approaches the resolution of the bar code generation system.

**Stacked Symbologies** Stacked bar codes were developed in the 1980s to encode more data in a smaller footprint than allowed by the then current linear bar codes. Today, the primary advantage to using a stacked symbology is to encode more data into a symbol that can still be read by a laser bar code scanner. While matrix codes are much more space efficient and offer increased data capacity than stacked 2D codes, matrix codes must be read with an image-based reader.[5] One significant drawback to stacked symbologies is their sensitivity to scanner tilt. To decode the symbol, the laser beam must be aligned so that the beam passes through the rows one at a time. The stacked symbology PDF417 provides the user with a little more tolerance. Since PDF417 has three different encoding schemes that read every three rows, the laser beam can pass through up to three rows at a time and still decode the symbol

**Code 49** was introduced in 1987 and was designed for labeling small objects. *Advantage:* It has a smaller footprint than most standard linear bar codes. *Disadvantage:* It has a maximum capacity of 49 alphanumeric characters and does not offer built-in error correction.

**Code 16K** was introduced a year after Code 49 and has an inverted Code 128 character set and has much more data capacity than Code 49. *Advantage:* It is much more compact than standard linear bar codes. *Disadvantage:* It is not widely used and does not offer built-in error correction.

.

**PDF417** symbology stands for Portable Data File and unlike the other stacked symbologies offers the robust Reed-Solomon form of error correction. This enables the symbol to sustain considerable damage and still be readable.PDF417 has three different encoding schemes that repeat every three rows. So, for example, the bar pattern for the character ‘‘A’’ in row one is not the same pattern for ‘‘A’’ in row 2. However, it is the same pattern in row 4. Since the encoding scheme repeats every three rows, the symbology can sustain a certain degree of scanner tilt and still be readable. *Advantage:* PDF417 has high data capacity and offers Reed-Solomon error correction, which makes it highly secure. It can also be read with a laser scanner in many applications. *Disadvantage:* Like other stacked symbologies, it is sensitive to scanner tilt and is not as space efficient as matrix codes

**RSS/Composite Symbology** The linear symbology RSS has a two-dimensional option called a Composite component. If additional information is needed, the Composite component can be attached to the top of the RSS symbol. The composite options are based on PDF417 and Micro-PDF417. *Advantage:* The composite component increases the data capacity of the RSS symbol. *Disadvantage:* Because the composite symbol is printed so small, the symbol’s sensitivity to scanner tilt becomes even more of a concern and can affect ease of use. An image-based reader is recommended for reading RSS/Composite symbols.

### 1.3.1 Advantages and disadvantages of Barcode Symbologies

| No | Barcode Symbologies | advantages | disadvantages |
|---|---|---|---|
| 1 | **UPC/EAN** | The UPC code has good data security and immunity to ink spread | It has a limited number of characters (numeric characters only) and is used for point-of-sale applications |
| 2 | **Code 39** | Code 39 offers good security and an alphanumeric character set. | Code 39 is not recommended for applications requiring long data identifiers. |
| 3 | **Code 128** | Its large character set combined with high density and mandatory check character maximizes data integrity. | Since Code 128 is a modular symbology, users must be aware that problems can arise when the narrow bar width approaches the resolution of the bar code generation system. |
| 4 | **Interleaved 2 of 5** | It is the highest density linear bar | It has very low security. The |

| | | | |
|---|---|---|---|
| | | code for numeric only strings of fewer than 10 characters. | symbology uses two bar patterns for start and stop instead of full characters, which can lead to truncation errors. |
| 5 | **Code 93** | It  always uses two check digits to ensure data security also the densest linear symbology | The symbology is not a standard symbology read by most scanners, and its varies in dimensions |
| 6 | **Code 49** | It has a smaller footprint than most standard linear bar codes. | It has a maximum capacity of 49 alphanumeric characters. |

| | | | |
|---|---|---|---|
| 7 | **Code 16K** | It is much more compact than standard linear bar codes. | It is not widely used and does not offer built-in error correction |
| 8 | **Pdf417** | PDF417 has high data capacity and offers Reed-Solomon error correction, which makes it highly secure. It can also be read with a laser scanner in many applications. | Like other stacked symbologies, it is sensitive to scanner tilt and is not as space efficient as matrix codes. |
| 9 | **MacroPdf417** | It can encode and decode files of unlimited size and has high error correction level | It cannot encode and decode audio,video and image files |

# 2. SYSTEM REQUIREMENTS & SPECIFICATION

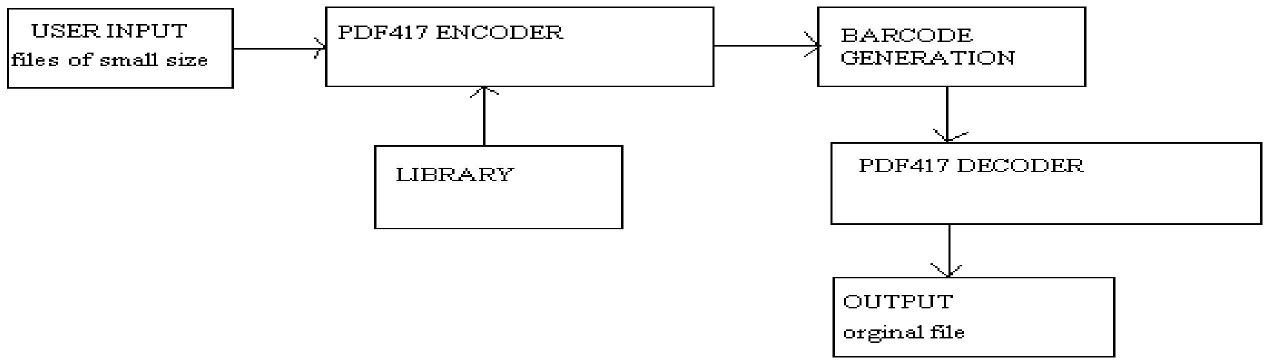
## 2.1 PROBLEM STATEMENT

### 2.1.1 EXISTING SYSTEM

**PDF417** is a multi-row, variable-length symbology with high data capacity and error-correction capability. PDF417 has some unique features which makes it the widely used 2D symbology. A PDF417 symbol can be read by linear scanners, laser scanners or two-dimensional scanners. PDF417 is capable of encoding more than 1100 bytes, 1800 text characters or 2710 digits..

In addition to features typical of two dimensional bar codes, PDF417's capabilities include:

- Linking. PDF417 symbols can link to other symbols which are scanned in sequence allowing even more data to be stored.
- User-specified dimensions. The user can decide how wide the narrowest vertical bar (X dimension) is, and how tall the rows are (Y dimension).
- Public domain format. Anyone can implement systems using this format without any license.



2.1.1.2 EXISTING SYSTEM

### 2.1.2 PROPOSED SYSTEM

Thus we propose Macro PDF417 which is capable of overcoming the disadvantages of PDF417 ie.it can able to encode and decode files of unlimited size. Macro PDF417 symbols differ from ordinary PDF417 symbols in that they contain additional control information in a zone called Macro PDF417 Control Block. Using Macro PDF417, large data files are split into several data segments and encoded into individual symbols. The Macro PDF417 decoder device uses the Control Block's information to reconstruct the file correctly, independent of symbol scanning order.

The Various Macro PDF417 Properties

- **Macro PDF Enable-**A Boolean indicating that this barcode is a part of  Macro pdf417 sequence.
- **Macro PDF File ID-**Assign a file ID to the MacroPDF barcode.Each barcode in the MacroPDF sequence  must have the same file ID assigned to it.Default is 0 valid options are 0-899.
- **Macro PDF Segment Index-**The index number of this MacroPDF Barcode in relation to the set.Default is 0;valid option are 0-99999;*each barcode in the MacroPDF sequence must have a unique segment index,starting at zero and incrementing thereafter by 1.
- **Macro PDF Last Segment-**A Boolean indicating that this is the final barcode in the MacroPDF sequence.

## 2.1.3 SYSTEM REQURIMENTS

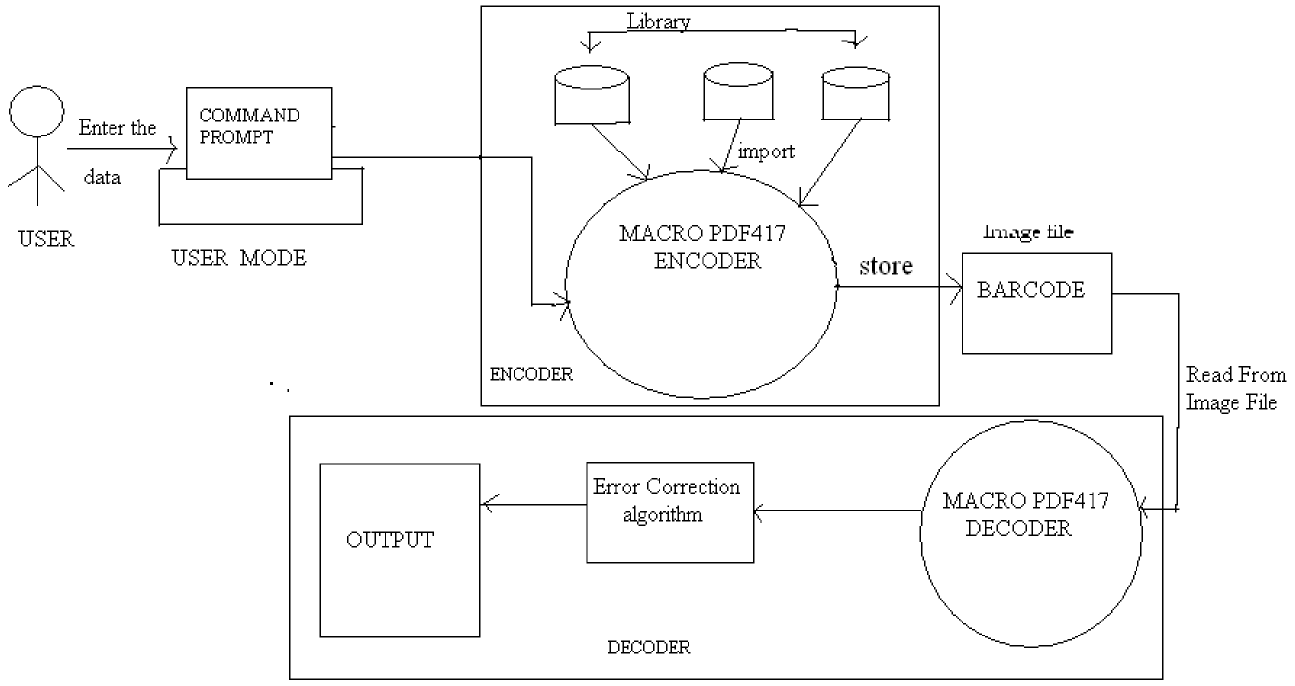
**HARWARE:**

| | |
|---|---|
| Processor | : pentium111 |
| Hard Disk | : 20 GB |
| RAM | : 128 MB |

**SOFTWARE:**

Core JAVA

## 2.2 OVERVIEW OF THE APPROACH



**2.2.1** Overall Architecture

# 3 DESIGN AND IMPLEMENTATION

## 3.1 DESIGN DETAILS

Modules:

- PDF 417 ENCODER
- MACRO PDF417 ENCODER
- MACRO PDF417 DECODER

**PDF 417 ENCODER:**

This module is to generate barcode for data using PDF417. The user has to enter data of limited size in the command prompt.The barcode generator generates the barcode using PDF library files.The generated barcode is displayed to user in the jpeg format.

**MACRO PDF417 ENCODER:**

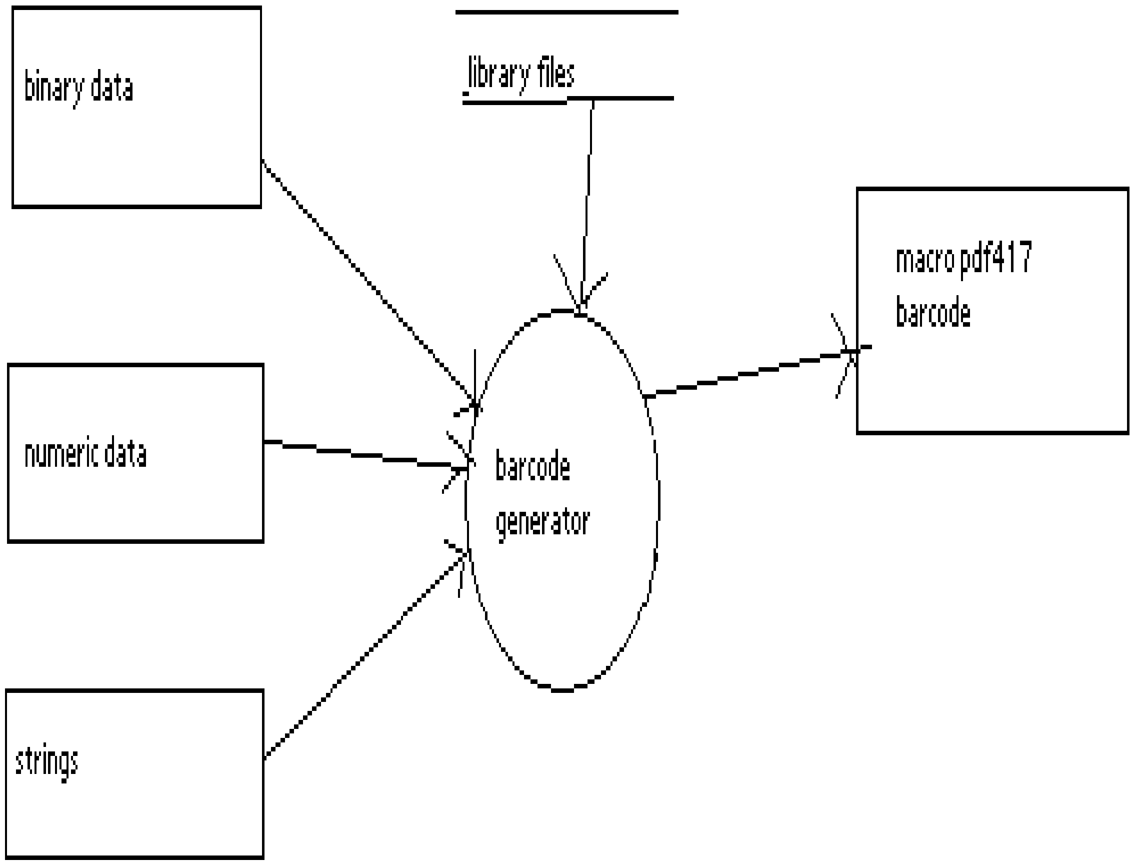
Large data files can be encoded into a series of linked PDF417 symbols using a standard methodology referred to as Macro PDF417 .This module is to generate barcode for large files using macro PDF417 which is an extension of PDF 417. The user has to enter alphanumeric characters or can browse it and submit it. The barcode encoder generates the barcode using Macro PDF library files.The generated barcode is displayed to user.

**MACRO PDF417 DECODER**:

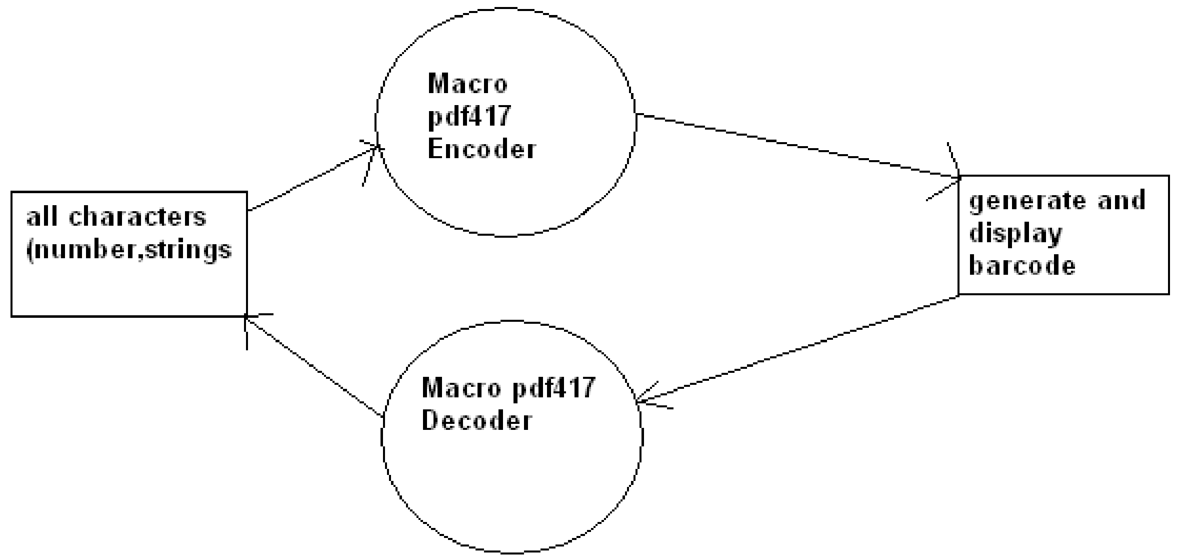
This module is to generate the given source file for barcode given. The user as to give barcode image as input and submit it. On submitting it, source file is decoded from the barcode image and displayed. The generated source file displayed to user is the secret information exchanged .

## 3.2 DATA FLOW DIAGRAMS

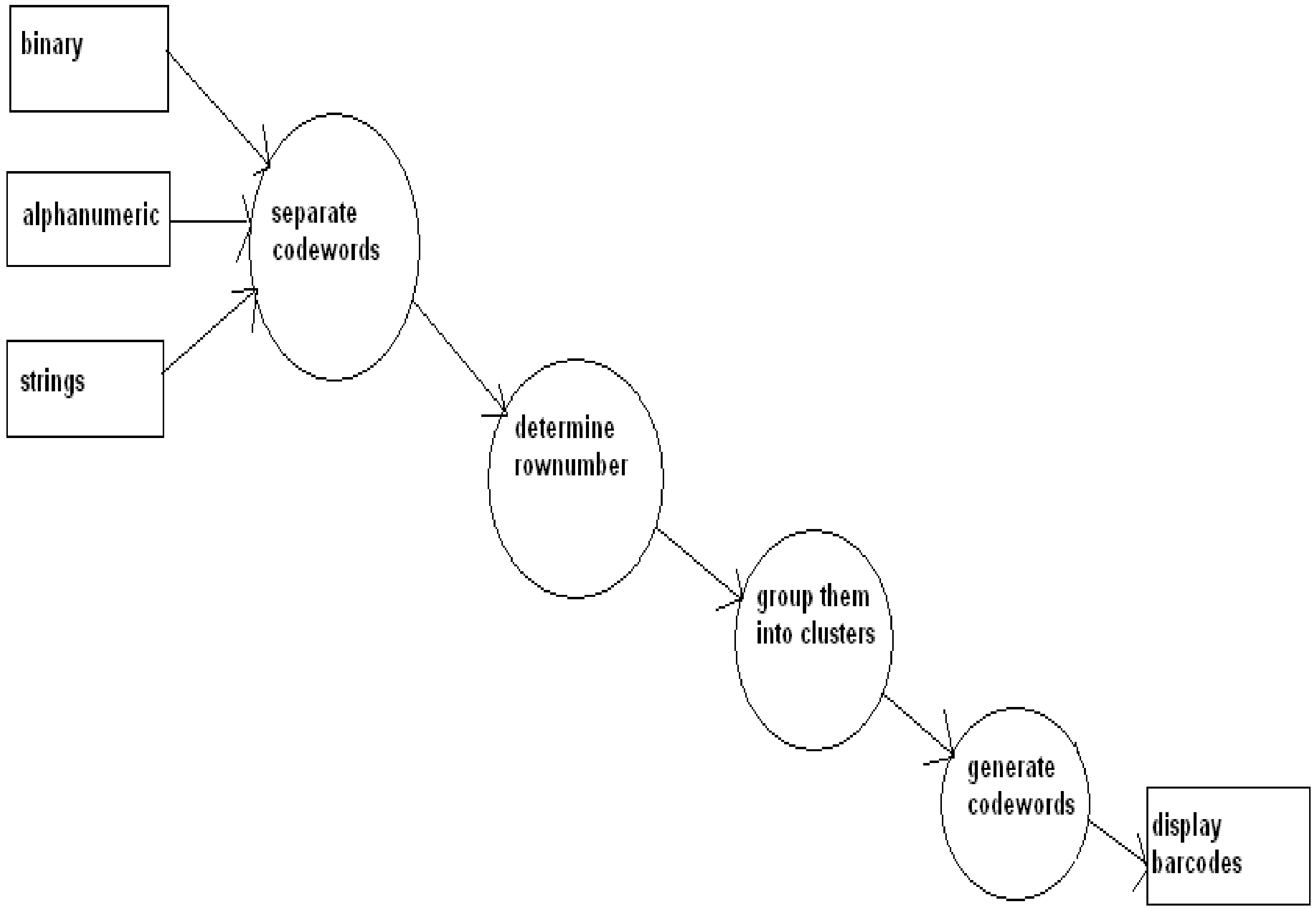
### DATA FLOW DIAGRAM – LEVEL0



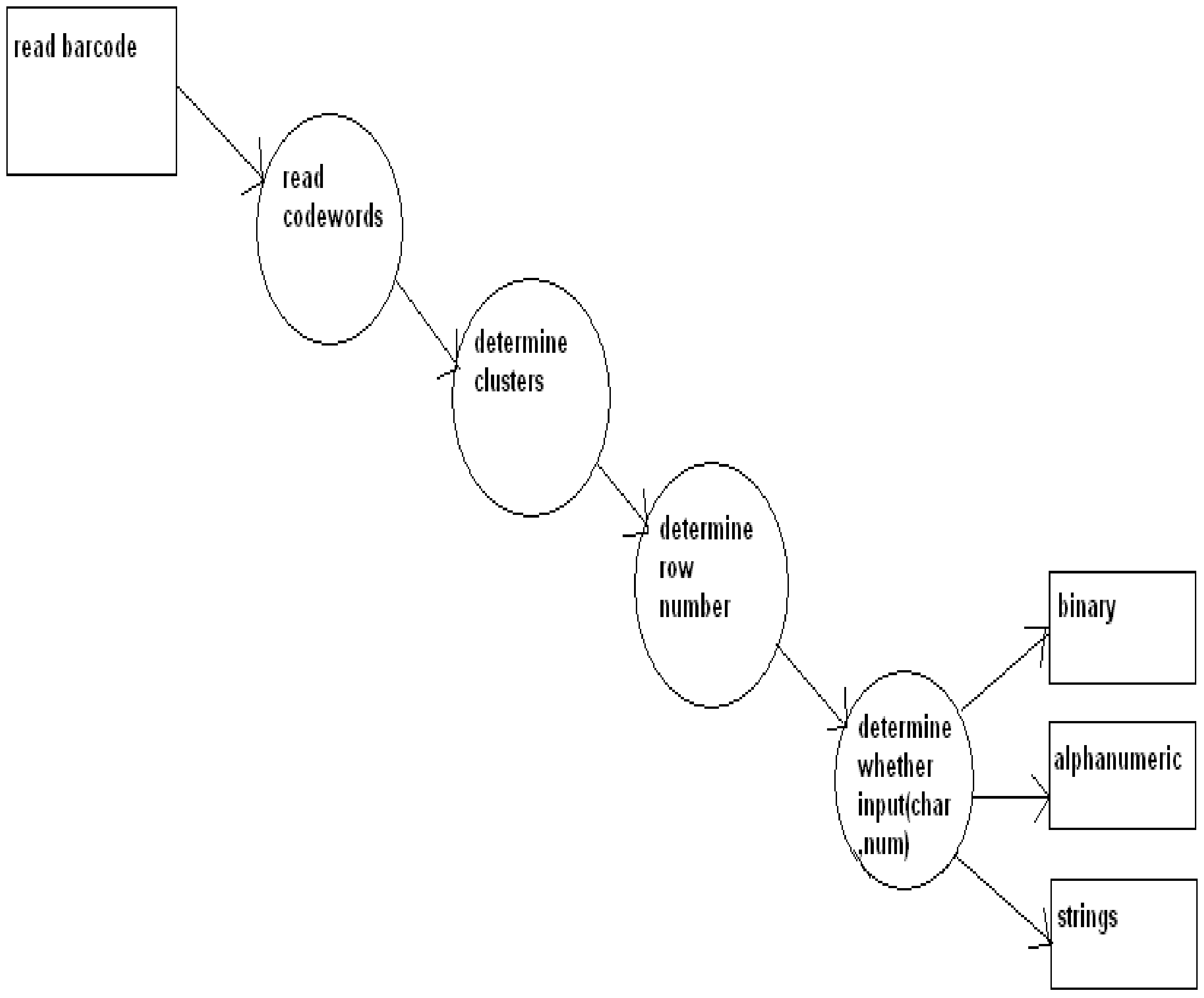
## LEVEL 1:


Macro pdf417 Encoder
all characters (number,strings
generate and display barcode
Macro pdf417 Decoder

## <u>LEVEL 2:Macro PDF417 Encoder</u>



## LEVEL 2:Macro PDF417 Decoder



## 3.3 IMPLEMENTATION DETAILS:

### PDF 417 ENCODER

#### Format

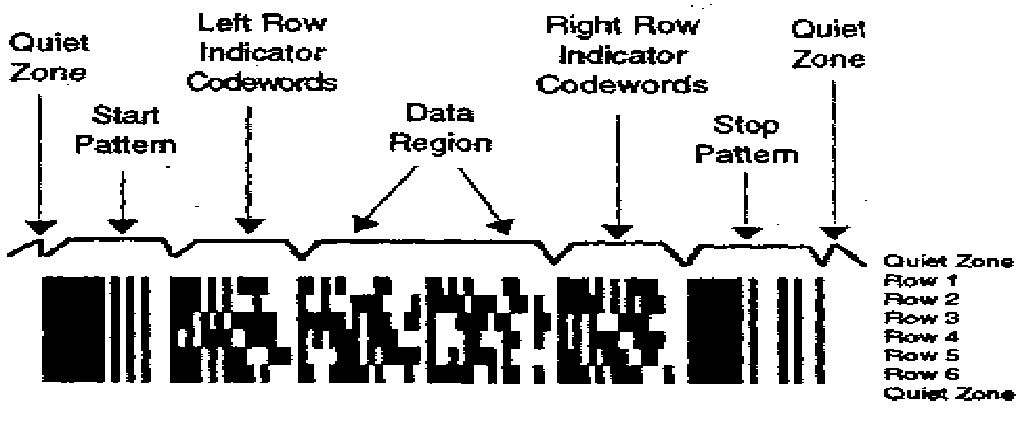
The PDF417 bar code (also called a **symbol**) consists of 3 to 90 rows, each of which is like a small linear bar code. Each row has:

- a **quiet zone**. This is a mandated minimum amount of white space before the bar code begins.
- a start pattern which identifies the format as PDF417. Every type of bar code symbology has a unique start and stop pattern.
- a "row left" codeword containing information about the row (such as row number and what error correction rate the row is using)
    - 1 - 30 data **codewords** : Codewords are a group of bars and spaces representing one or more numbers, letters, or other symbols.
    - All rows have the same number of codewords.
    - Every codeword contains four bars and four spaces (where the 4 in the name comes from).
    - The total width of a codeword is 17 times the width of the narrowest allowed vertical bar (the X dimension). This is where the 17 in the name comes from.
    - Each codeword starts with a bar and ends with a space.
    - There are 929 codewords to choose from, 900 for data, and 29 for special functions.
    - Each codeword is printed using one of three distinct **clusters**:

  - A cluster is a bar-space pattern for each of the 929 codewords
  - No bar-space pattern is repeated between clusters
  - The row number determines which cluster to use
  - The cluster is the same for all codewords in a row.
  - The purpose of clusters is to determine which row (mod 3) the codeword is in, allowing the scan to be skewed from the horizontal. For instance, the scan might start on row 6 at the start of the row, and be on row 10 at the end.
- A "row right" codeword with more information about the row.
- A stop pattern.
- A quiet zone.

**Symbol Structure**

A typical PDF417 symbol contains 3 to 90 rows. Each row consists of



### 3.3.1 Symbol Structure

- Leading quiet zone
- Start Pattern

- Left row indicator symbol character
- 1 to 30 data symbol characters
- Right row indicator symbol character
- Stop pattern.
- Trailing quiet zone

## MACRO PDF417 ENCODER

Macro PDF417 symbols differ from ordinary PDF417 symbols in that they contain additional control information in a zone called Macro PDF417 Control Block.The properties of macro PDF 417 barcode can be set as follows:

1. Set the **data** property with the value to encode. Type is **String**. Servlet URL Parameter: "Data".
2. Set the **dataMode** property. Default is PDF417.Text.
   - **PDF417.AUTO (0):** Barcode Library will use the different compaction modes and switching procedures automatically.
   - **PDF417.MODE_TEXT (1) (default):** It allows encoding all printable ASCII characters, i.e. values from 32 to 126 inclusive in accordance with ISO/IEC 646, as well as selected control characters such as TAB (horizontal tab ASCII 9), LF (NL line feed, new line ASCII 10) and CR (carriage return ASCII 13)

- **PDF417.MODE_BYTE (2):** It allows encoding all 256 possible 8-bit byte values. This includes all ASCII characters value from 0 to 127 inclusive and provides for international character set support
- **PDF417.MODE_NUMERIC (3):** It allows encoding numeric data.

Servlet URL Parameter: "DataMode". Sample: &DataMode=1

3. Set the **processTilde** property to true, if you want use the tilde character "~" to specify special characters in the encoding data. Default is false.
   Format of the tilde:
   - ~NNN: is used to represent the ASCII character with the value of NNN. NNN is from 000 - 255.
   - ~6NNNNN: is used to represent the Unicode. NNNNN is from 00000 - 65535.
   - ~7NNNNNN: is used to specify the Extended Channel Interpretations and NNNNNN is a value between 000000 and 999999.
   - ~rp: is used only at the very beginning of the symbol for the reader programming purpose.

   Servlet Parameter: "ProcessTilde". Value: "t" (true), "f" (false). Sample: &ProcessTilde=t

4. Set the **ecl** property. PDF417 Error Correction Level. Default is PDF417.ECL_2 (2).
   Servlet URL Parameter: "ECL". Sample: &ECL=2
5. Set the **truncated** property to true, if you want truncated PDF417. Default is false.
   truncated PDF417 may be used where space considerations are a primary concern and symbol damage is unlikely.
   Servlet URL Parameter: "Truncated". Value: "t" (true), "f" (false). Sample: &Truncated=t
6. Set the **column** property value.
   Number of columns. The value range is from 1 to 30. The default is 5. Increase this value, if your data size is large.
   Servlet URL Parameter: "ColumnCount". Sample: &ColumnCount=5
7. Set the **row** property value.
   The number of rows for PDF417. The value range is from 3 to 90. The default is 3.
   Servlet URL Parameter: "RowCount". Sample: &RowCount=3
8. Macro PDF 417
   PDF 417 can be divided into multiple data areas, also called Macro PDF 417. Conversely, information stored in multiple PDF417 symbols can be reconstructed as single data symbols.
   - Set **isMacro** property to true, then Macro PDF417 is enabled.
     Servlet Parameter: "IsMacro".
   - Set **macroSegmentCount** property to the number of total symbols which make the sequence.
     Servlet Parameter: "MacroSegmentCount".

- Set **macroSegmentIndex** property to the position of current symbol in the secuence (Start with 0).
  Servlet Parameter: "MacroSegmentIndex".
- Set **macroFileIndex** property to be identified to the same file .
  Servlet Parameter: "MacroFileIndex".

9. Setting up barcode image size:
   - Set property **uom** (Unit of Measure) for properties X, Y, leftMargin, rightMargin, topMargin and bottomMargin.
     Default is Linear.UOM_PIXEL (0). Valid values are Linear.UOM_PIXEL (0), Linear.UOM_CM (1), Linear.UOM_Inch (2).
     Servlet URL Parameter: "UOM". Value: 0 (pixel), 1 (cm), 2 (inch). Sample: &UOM=0
   - Set the **X** (for barcode module width) and **barRatio** (barcode module height = X * barRatio) properties.
     X type is float, default is 1.
     barRatio type is int, default is 5.
     Servlet URL Parameter: "X", "BarRatio".
   - Set the **leftMargin**, **rightMargin**, **topMargin** and **bottomMargin** properties, and types are all **float**.
     Default values are 0 for all 4 margin settings.
     Servlet URL Parameter: "LeftMargin", "RightMargin", "TopMargin", "BottomMargin". Sample: &LeftMargin=0
   - Set the **resolution** property (Value is expressed in DPI - Dots per inch).
     Default is 72 dpi.

Servlet URL Parameter: "Resolution". Sample: &Resolution=72

10. With **rotate** property, you can display barcode horizontally or vertically.
    Value can be
    - 0 (Linear.ANGLE_0),
    - 1 (Linear.ANGLE_90),
    - 2 (Linear.ANGLE_180),
    - 3 (Linear.ANGLE_270)
    - Servlet URL Parameter: "Rotate".

## <u>**Macro PDF417 Example**</u>

First Segment:

MacroPDFEnable = True

MacroPDFFileID = 237

MacroPDFSegmentIndex* = 0

MacroPDFLastSegment = False

Second Segment:

MacroPDFEnable = True

MacroPDFFileID = 237

MacroPDFSegmentIndex* = 1

## Description of encoding algorithm

The amount of data compaction will vary depending upon the type of data that is being encoded and the error correction level chosen. If the scanner supports it, smaller symbols may be obtained by performing any of the following:

1. Use TEXT compaction mode.
2. Decrease the error correction level to 2.
3. Increase the number of columns to the largest possible size.
4. Decrease the X dimension to the smallest possible size the scanner will read.
5. Turn on truncation, which decreases the symbol size by two columns.
6. Choose an X to Y ratio of 1:3.

## MACRO PDF417 DECODER

In this module the barcode which is in the svg format is given as input and the macro pdf417 decoder must use the library files to decode and get the orginal file back.

The "edge to similar edge" estimation method is employed to check whether the detected bar-space pattern is correct. Below shows the pattern information obtained by scanning the warped image given below

**31112144 1233113** iiimx **14141411** 131114~**m mii 1 2 2 im3**

**5212311221112~361142111 11413133 1?psllu li32tm 125511114**

**1142W22114?41142!32122 Pit3111 33122114 3111233522114722**

**1 ~%1 3 2 1 1 4 ~ 2 5 1 2 1 1 1 41121 42521 312212% 351L2Z 1221m3**

**31132511124211425221133l2 43111412 PIE412 28LWll lL%%Jl**

The bar-space patterns depicted in can be converted to the encoded codewords by using the lookup table specified . The codewords are checked whether there are any errors through Reed-Solomon error correction algorithm. The codeword sequence after applying error correction algorithm are as follows.

Error Correct Level: **3**

I errors corrected

194 **453** 172 **824** 52n **547 435 aoo 709 359**

**746 467 441 243 138 800 544 528 802 258**

**236** -1 **3 784 18 746 364 577 423 794** 179

Given the error-corrected codewords, they *are* decoded in the manner as specified in the message the error corrected codewords .

## 4 RESULTS & DISCUSSION

### 4.1 TEST CASE

| **S.NO** | **INPUT** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** |
|---|---|---|---|
| 1 | Numeric file |  | TRUE |
| 2 | Text file |  | TRUE |
| 3 | Mixed file type |  | TRUE |
| 4 |  | Appropriate file content | TRUE |

## 4.2 DISCUSSION:

**Comparsion of different barcodes**

| no | Parameters | Interleaved 2 of 5 | Code 128 | Pdf417 | MacroPdf417 |
|---|---|---|---|---|---|
| 1 | speed | Very slow | It is fast compared to the traditional symbologies | It is fast when compared with code 128 | It is very fast |
| 2 | capacity | Only limited size of numeric files can be encoded | It can encode as many as 2725 data characters in a single bar code. | It store up to about 1,800 printable ASCII characters or 1,100 binary characters per symbol. | It can encode and decode files of unlimited size |
| 3 | dimensions | Numeric-only barcodes | Alpha-numeric barcodes | 2-Dimensional barcodes | 2-Dimensional barcodes |
| 4 | Error correction level | It does nt have any error correction level | Calculation of checksum is little tricky | LEVEL 0 (no error correction) to LEVEL 8 (the maximum error correction | It uses the same error correction level used in pdf417 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | level) | |
| 5 | format | 1234 | ABCxyz#$%15z | | |
| 6 | Advantages | It is the highest density linear bar code for numeric only strings of fewer than 10 characters. | Its large character set combined with high density and mandatory check character maximizes data integrity. | PDF417 has high data capacity and offers Reed-Solomon error correction, which makes it highly secure. | It can encode and decode files of unlimited size and has high error correction level |

## 5. CONCLUSION AND FUTURE WORK

Thus the proposed system  Macro PDF 417 encoder is used to encode large files into barcode and the decoder reads the barcode to get the files encoded in the barcode image.This is used in E-GOVERNANCE  application  to transfer secret files by encoding it to a barcode image. Macro PDF 417 can be extended to 3D barcode symbology to encode audio and video files and also image files which is left for future research.

# 6 APPENDICES

## 6.1 APPENDIX 1:

/* **main class for macro pdf417 encoder***/

```
public class Main {

 private static final String[] APP_HEADER = {

 "Barcode4J command-line application, Version " + getVersion(), ""};

  public static PrintStream stdout = System.out;

  public static PrintStream stderr = System.err;

   private static ExitHandler exitHandler = new DefaultExitHandler();

  private Options options;

 private boolean headerPrinted = false;

 private Logger log;

    public static void main(String[] args) {

        Main app = new Main();

        app.handleCommandLine(args);

    }

 public static void setExitHandler(ExitHandler handler) {
    exitHandler = handler;

}

public void handleCommandLine(String[] args) {

    CommandLine cl;

    String[] msg;

    try {

            CommandLineParser clp = new PosixParser();

               cl = clp.parse(getOptions(), args);

                  msg = cl.getArgs();

        if (msg.length == 0) {

            throw new ParseException("No message");

        }

        if (msg.length > 1) {

            throw new ParseException("Too many parameters: " msg.length);

        }

    } catch (MissingOptionException moe) {

        printHelp(new PrintWriter(stdout));
```

```
            exitHandler.failureExit(this,   "Bad command line. Missing option: "
+ moe.getMessage(), null, -2);

            return;

        }

catch (ParseException pe) {

            printHelp(new PrintWriter(stdout));

            //pe.printStackTrace();

            exitHandler.failureExit(this,

                "Bad command line: " + pe.getMessage(), null, -2);

            return; //never reached

        }

  try {

            OutputStream out;

if (!cl.hasOption("o")) {

  log = new
AdvancedConsoleLogger(AdvancedConsoleLogger.LEVEL_ERROR,
false, stderr, stderr);

                printAppHeader();
```

```
   out = stdout;
} else {
      int logLevel = AdvancedConsoleLogger.LEVEL_INFO;
   if (cl.hasOption('v')) {
      logLevel = AdvancedConsoleLogger.LEVEL_DEBUG;
   }
   log = new AdvancedConsoleLogger(logLevel, false, stdout,stderr);
   printAppHeader();
   File outFile = new File(cl.getOptionValue("o"));
   if (log.isDebugEnabled()) {
      log.debug("Output to: " + outFile.getCanonicalPath());
   } out = new java.io.FileOutputStream(outFile);
}

log.debug("Message: " + msg[0]);
```

```
            String format = MimeTypes.expandFormat(cl.getOptionValue("f",
MimeTypes.MIME_SVG));

            int orientation = 0;

            log.info("Generating " + format + "...");

            BarcodeUtil util = BarcodeUtil.getInstance();

            BarcodeGenerator gen = util.createBarcodeGenerator(

                    getConfiguration(cl));

             if (MimeTypes.MIME_SVG.equals(format)) {



                SVGCanvasProvider svg = new
SVGCanvasProvider(false,orientation);

                gen.generateBarcode(svg, msg[0]);

     try {

                    TransformerFactory factory =
TransformerFactory.newInstance();

                    Transformer trans = factory.newTransformer();

                    Source src = new javax.xml.transform.dom.DOMSource(

                        svg.getDOMFragment());
                    Result res = new javax.xml.transform.stream.StreamResult(out);

                    trans.transform(src, res);

                } catch (TransformerException te) {

                    exitHandler.failureExit(this, "XML/XSLT library error", te, -6);

                }

            }

              else {

                int dpi = Integer.parseInt(cl.getOptionValue('d', "300"));

                log.debug("Resolution: " + dpi + "dpi");

                BitmapCanvasProvider bitmap;

                if (cl.hasOption("bw"))

                 {

                    log.debug("Black/white image (1-bit)");

                    bitmap = new BitmapCanvasProvider(out,

                        format, dpi, BufferedImage.TYPE_BYTE_BINARY, false,
orientation);

                }
```

```
                else
                {
                    log.debug("Grayscale image (8-bit) with anti-aliasing");
                    bitmap = new BitmapCanvasProvider(out,
                        format, dpi, BufferedImage.TYPE_BYTE_GRAY, true,
orientation);
                }
                gen.generateBarcode(bitmap, msg[0]);
                bitmap.finish();
            }
    out.close();
            log.info("done.");
            exitHandler.successfulExit(this);
        } catch (IOException ioe) {
            exitHandler.failureExit(this,
                "Error writing output file: " + ioe.getMessage(), null, -5);
        } catch (ConfigurationException ce) {
            exitHandler.failureExit(this,
                "Configuration problem: " + ce.getMessage(), ce, -6);
        } catch (BarcodeException be) {
            exitHandler.failureExit(this,
                "Error generating the barcode", be, -3);
        }
    }

    private Options getOptions() {
        if (options == null) {
            this.options = new Options();
            Option opt;

            this.options.addOption(OptionBuilder
                .withLongOpt("verbose")
                .withDescription("enable debug output")
                .create('v'));
```

```
this.options.addOption(OptionBuilder
    .withLongOpt("output")
    .withArgName("file")
    .hasArg()
    .withDescription("the output filename")
    .create('o'));


 //Group: config file/barcode type
 OptionGroup group = new OptionGroup();
 group.setRequired(true);
 group.addOption(OptionBuilder
    .withArgName("file")
    .withLongOpt("config")
    .hasArg()
    .withDescription("the config file")
    .create('c'));
 group.addOption(OptionBuilder
    .withArgName("name")

    .withLongOpt("symbol")

    .hasArg()

    .withDescription("the barcode symbology to select "

        + "(default settings, use -c if you want to customize)")

    .create('s'));

this.options.addOptionGroup(group);



//Output format type

this.options.addOption(OptionBuilder

    .withArgName("format")

    .withLongOpt("format")

    .hasArg()

    .withDescription("the output format: MIME type or file "

        + "extension\n"

        + "Default: " + MimeTypes.MIME_SVG + " (SVG)")

    .create('f'));
```

```
}



        //Bitmap-specific options

        this.options.addOption(OptionBuilder

            .withArgName("integer")

            .withLongOpt("dpi")

            .hasArg()

            .withDescription("(for bitmaps) the image resolution in dpi\n"

                + "Default: 300")

            .create('d'));

        this.options.addOption(OptionBuilder

            .withLongOpt("bw")

            .withDescription("(for bitmaps) create monochrome (1-bit) "

                + "image instead of grayscale (8-bit)")

            .create());

    }

    return this.options;

}
```

```
private Configuration getConfiguration(CommandLine cl) {
    if (cl.hasOption("s")) {
        String sym = cl.getOptionValue("s");
        DefaultConfiguration cfg = new DefaultConfiguration("cfg");
        DefaultConfiguration child = new DefaultConfiguration(sym);
        cfg.addChild(child);
        return cfg;
    }
    if (cl.hasOption("c")) {
        try {
            String filename = cl.getOptionValue("c");
            File cfgFile = new File(filename);
            if (!cfgFile.exists() || !cfgFile.isFile()) {
                throw new FileNotFoundException(
                    "Config file not found: " + cfgFile);
            }
```

```
                log.info("Using configuration: " + cfgFile);



                DefaultConfigurationBuilder builder = new
DefaultConfigurationBuilder();

                return builder.buildFromFile(cfgFile);

            } catch (Exception e) {

                exitHandler.failureExit(this,

                    "Error reading configuration file: " + e.getMessage(), null, -3);

            }

        }

        return new DefaultConfiguration("cfg");

    }



    /** @return the Barcode4J version */

    public static String getVersion() {

        String version = null;

        Package jarinfo = Main.class.getPackage();
    if (jarinfo != null) {
        version = jarinfo.getImplementationVersion();
    }
    if (version == null) {
      version = "DEV";
    }
    return version;
}


/**
 * Prints the application header on the console. Ensures that this is only
 * done once.
 */
public void printAppHeader() {
    if (!headerPrinted) {
        if (log != null) {
            for (int i = 0; i < APP_HEADER.length; i++) {
                log.info(APP_HEADER[i]);

            }

        } else {

            for (int i = 0; i < APP_HEADER.length; i++) {

                stdout.println(APP_HEADER[i]);

            }

        }

        headerPrinted = true;

    }

}



private void printHelp(PrintWriter writer) {

    printAppHeader()

//Get a list of additional supported MIME types

    Set knownMimes = new java.util.HashSet();

    knownMimes.add(null);

    knownMimes.add("");
```

```
        knownMimes.add(MimeTypes.MIME_PNG);

        knownMimes.add("image/png");

        knownMimes.add(MimeTypes.MIME_JPEG);

        knownMimes.add(MimeTypes.MIME_TIFF);

        knownMimes.add(MimeTypes.MIME_GIF);

        Set additionalMimes =
BitmapEncoderRegistry.getSupportedMIMETypes();

        additionalMimes.removeAll(knownMimes);

 HelpFormatter help = new HelpFormatter();

        final String unavailable = " (unavailable)";

        help.printHelp(writer, HelpFormatter.DEFAULT_WIDTH,

            "java -jar barcode4j.jar "

                + "[-v] [[-s <symbology>]|[-c <cfg-file>]] [-f <format>] "

                + "[-d <dpi>] [-bw] [-o <file>] <message>",

            null,

            getOptions(),

            HelpFormatter.DEFAULT_LEFT_PAD,
HelpFormatter.DEFAULT_DESC_PAD,
            "Valid output formats:"
                + "\nSVG: " + MimeTypes.MIME_SVG + "
               + "\nJPEG: " + MimeTypes.MIME_JPEG + ", jpeg, jpg"
                  + (BitmapEncoderRegistry.supports(MimeTypes.MIME_JPEG)
                    ? "" : unavailable)
                + (additionalMimes.size() > 0
                  ? "\nAdditional supported formats:\n" + additional : "")
                }
```

**/*Main program of decoder*/**

```
public final class CommandLineRunner {


 private CommandLineRunner() {
 }


 public static void main(String[] args) throws Exception {
  if (args == null || args.length == 0) {
   printUsage();
   return;
  }

  boolean tryHarder = false;
  boolean pureBarcode = false;
  boolean productsOnly = false;
  boolean dumpResults = false;
  boolean dumpBlackPoint = false;
  for (String arg : args) {
   if ("--try_harder".equals(arg)) {
     tryHarder = true;
   } else if ("--pure_barcode".equals(arg)) {
     pureBarcode = true;
   } else if ("--products_only".equals(arg)) {
     productsOnly = true;
   } else if ("--dump_results".equals(arg)) {
     dumpResults = true;
```

```
      } else if ("--dump_black_point".equals(arg)) {

        dumpBlackPoint = true;

      } else if (arg.startsWith("-")) {

        System.err.println("Unknown command line option " + arg);

        printUsage();

        return;

      }

    }



    Hashtable<DecodeHintType, Object> hints = buildHints(tryHarder,
pureBarcode, productsOnly);

    for (String arg : args) {

      if (!arg.startsWith("--")) {

        decodeOneArgument(arg, hints, dumpResults, dumpBlackPoint);

      }

    }

  }
```

```
}

  private static Hashtable<DecodeHintType, Object> buildHints(boolean
tryHarder,
                                                              boolean pureBarcode,
                                                              boolean productsOnly) {
    Hashtable<DecodeHintType, Object> hints = new
Hashtable<DecodeHintType, Object>(3);
    Vector<BarcodeFormat> vector = new Vector<BarcodeFormat>(8);
     vector.addElement(BarcodeFormat.PDF417);
    }
    hints.put(DecodeHintType.POSSIBLE_FORMATS, vector);
    if (tryHarder) {
      hints.put(DecodeHintType.TRY_HARDER, Boolean.TRUE);
    }
    if (pureBarcode) {
      hints.put(DecodeHintType.PURE_BARCODE, Boolean.TRUE);
    }
    return hints;

  }



  private static void printUsage() {

    System.err.println("Decode barcode images using the ZXing library\n");

    System.err.println("usage: CommandLineRunner { file | dir | url } [
options ]");

    System.err.println("  --try_harder: Use the TRY_HARDER hint, default is
normal (mobile) mode");

        System.err.println("  --dump_results: Write the decoded contents to
input.txt");

    System.err.println("  --dump_black_point: Compare black point
algorithms as input.mono.png");

  }



  private static void decodeOneArgument(String argument,
Hashtable<DecodeHintType, Object> hints,

      boolean dumpResults, boolean dumpBlackPoint) throws IOException,

      URISyntaxException {
```

```
    File inputFile = new File(argument);

    if (inputFile.exists()) {

      if (inputFile.isDirectory()) {

        int successful = 0;

        int total = 0;

        for (File input : inputFile.listFiles()) {

          String filename = input.getName().toLowerCase();

          // Skip hidden files and text files (the latter is found in the blackbox
tests).

          if (filename.startsWith(".") || filename.endsWith(".txt")) {

            continue;

          }

          // Skip the results of dumping the black point.

          if (filename.contains(".mono.png")) {

            continue;

          }
          Result result = decode(input.toURI(), hints, dumpBlackPoint);
          if (result != null) {
            successful++;
            if (dumpResults) {
              dumpResult(input, result);
            }
          }
          total++;
        }
        System.out.println("\nDecoded " + successful + " files out of " + total +
            " successfully (" + (successful * 100 / total) + "%)\n");
      } else {
        Result result = decode(inputFile.toURI(), hints, dumpBlackPoint);
        if (dumpResults) {
          dumpResult(inputFile, result);
        }
      }
```

```
    } else {

      decode(new URI(argument), hints, dumpBlackPoint);

    }

  }



  private static void dumpResult(File input, Result result) throws
IOException {

    String name = input.getAbsolutePath();

    int pos = name.lastIndexOf('.');

    if (pos > 0) {

      name = name.substring(0, pos);

    }

    File dump = new File(name + ".txt");

    writeStringToFile(result.getText(), dump);

  }



  private static void writeStringToFile(String value, File file) throws
IOException {
    Writer out = new OutputStreamWriter(new FileOutputStream(file),
Charset.forName("UTF8"));

    try {

      out.write(value);

    } finally {

      out.close();

    }

  }



  private static Result decode(URI uri, Hashtable<DecodeHintType, Object>
hints,

      boolean dumpBlackPoint) throws IOException {

    BufferedImage image;

    try {

      image = ImageIO.read(uri.toURL());

    } catch (IllegalArgumentException iae) {

      throw new FileNotFoundException("Resource not found: " + uri);

    }
    if (image == null) {
      System.err.println(uri.toString() + ": Could not load image");
      return null;
    }
    try {
      LuminanceSource source = new
BufferedImageLuminanceSource(image);
      BinaryBitmap bitmap = new BinaryBitmap(new
HybridBinarizer(source));
      if (dumpBlackPoint) {
        dumpBlackPoint(uri, image, bitmap);
      }
      Result result = new MultiFormatReader().decode(bitmap, hints);
      ParsedResult parsedResult = ResultParser.parseResult(result);
      System.out.println(uri.toString() + " (format: " +
result.getBarcodeFormat() +
          ", type: " + parsedResult.getType() + "):\nRaw result:\n" +
result.getText() +
          "\nParsed result:\n" + parsedResult.getDisplayResult());
      return result;

    } catch (NotFoundException nfe) {

      System.out.println(uri.toString() + ": No barcode found");

      return null;

    } finally {



      //System.out.println("Threw " +
ReaderException.getExceptionCountAndReset() + " exceptions");

      //System.out.println("Throwers:\n" +
ReaderException.getThrowersAndReset());

    }

  }

 private static void dumpBlackPoint(URI uri, BufferedImage image,
BinaryBitmap bitmap) {

    String inputName = uri.getPath();

    if (inputName.contains(".mono.png")) {

      return;

    }
    int width = bitmap.getWidth();

    int height = bitmap.getHeight();

    int stride = width * 3;

    int[] pixels = new int[stride * height]

  int[] argb = new int[width];

    for (int y = 0; y < height; y++) {

      image.getRGB(0, y, width, 1, argb, 0, width);

      System.arraycopy(argb, 0, pixels, y * stride, width);

    }

 BitArray row = new BitArray(width);

    for (int y = 0; y < height; y++) {

      try {

        row = bitmap.getBlackRow(y, row);

      } catch (NotFoundException nfe) {



        int offset = y * stride + width;
      for (int x = 0; x < width; x++) {
        pixels[offset + x] = 0xffff0000;
      }
      continue;
    }

    int offset = y * stride + width;
    for (int x = 0; x < width; x++) {
      if (row.get(x)) {
        pixels[offset + x] = 0xff000000;
      } else {
        pixels[offset + x] = 0xffffffff;
      }
    }
 }
```

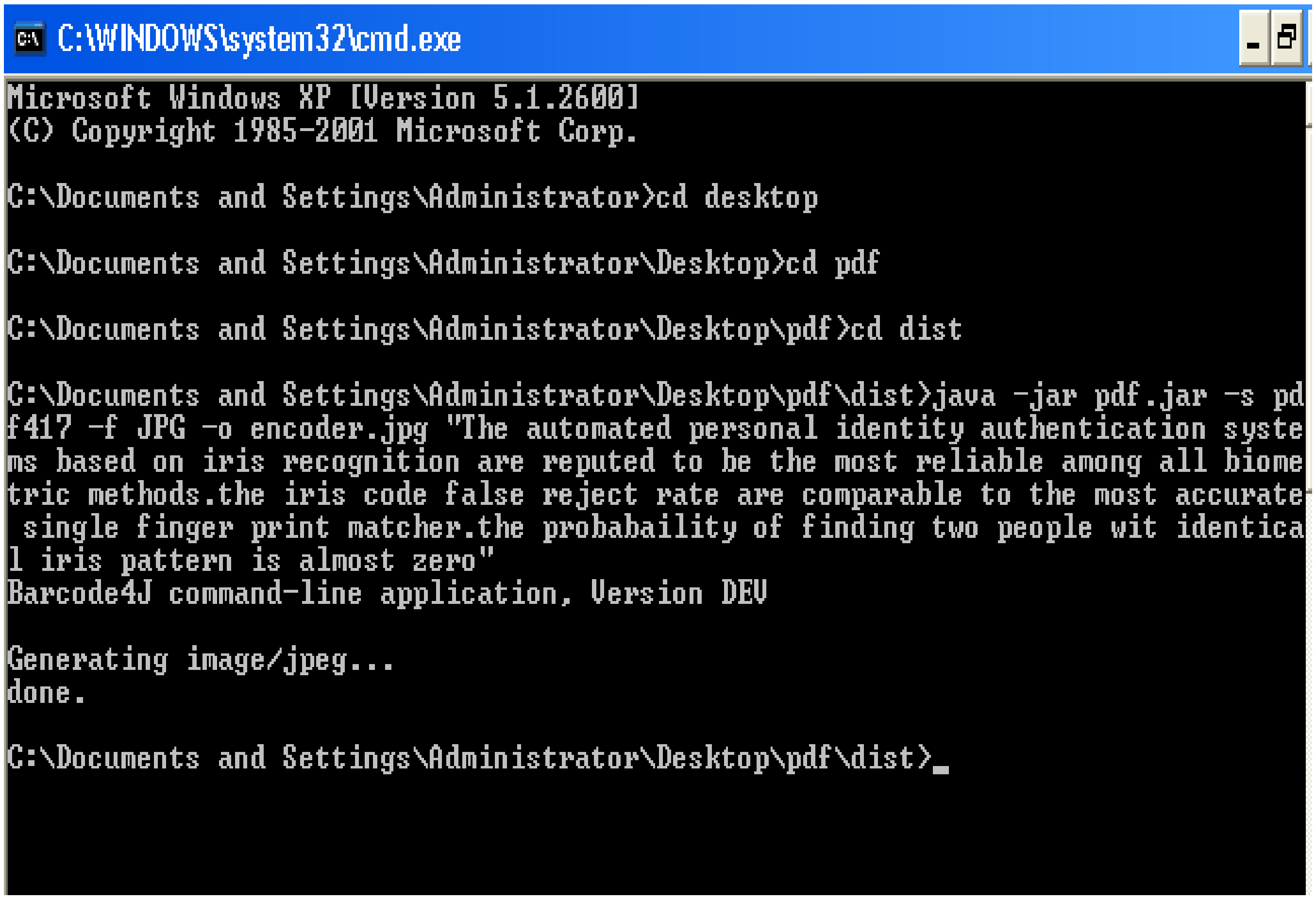
```
try {
  for (int y = 0; y < height; y++) {
    BitMatrix matrix = bitmap.getBlackMatrix();
    int offset = y * stride + width * 2;
    for (int x = 0; x < width; x++) {
      if (matrix.get(x, y)) {
        pixels[offset + x] = 0xff000000;
      } else {
        pixels[offset + x] = 0xffffffff;
      }
    }
  }
} catch (NotFoundException nfe) {
}
```

```
    BufferedImage result = new BufferedImage(stride, height,
BufferedImage.TYPE_INT_ARGB);

    result.setRGB(0, 0, stride, height, pixels, 0, stride);




    String resultName = inputName;

    if ("http".equals(uri.getScheme())) {

      int pos = resultName.lastIndexOf('/');

      if (pos > 0) {

        resultName = '.' + resultName.substring(pos);

      }

    }

    int pos = resultName.lastIndexOf('.');

    if (pos > 0) {

      resultName = resultName.substring(0, pos);

    }

    resultName += ".mono.png";
    OutputStream outStream = null;

    try {

      outStream = new FileOutputStream(resultName);

      ImageIO.write(result, "png", outStream);

    } catch (FileNotFoundException e) {

      System.err.println("Could not create " + resultName);

    } catch (IOException e) {

      System.err.println("Could not write to " + resultName);

    } finally {

      try {

        if (outStream != null) {

          outStream.close();

        }

      } catch (IOException ioe) {

        // continue

      }  }

  }}
```

## 6.2 APPENDIX 2

## SNAPSHOT OF THE ENCODER:

## EXECUTING THE JAR FILE IN COMMAND PROMPT

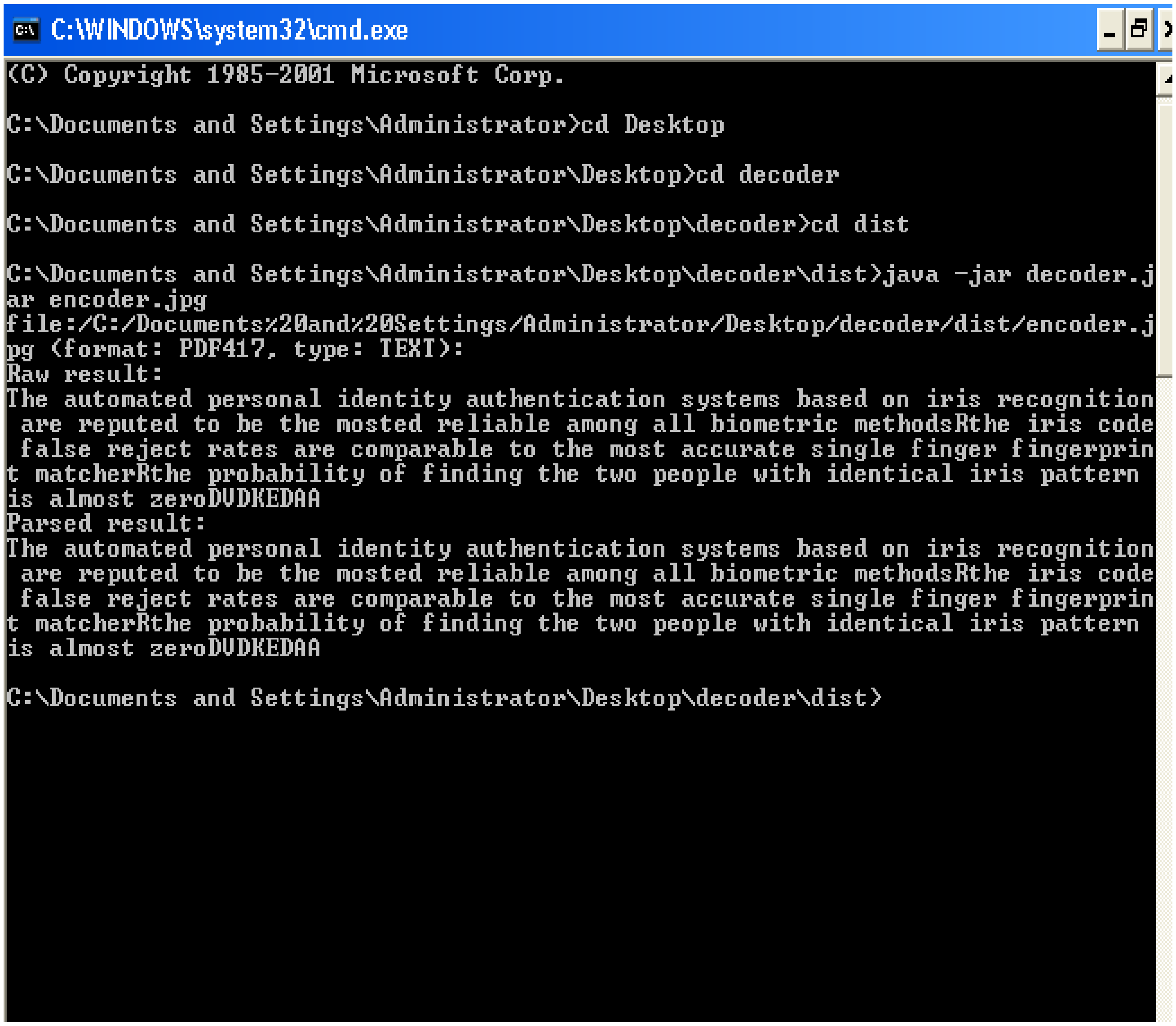


```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd desktop

C:\Documents and Settings\Administrator\Desktop>cd pdf

C:\Documents and Settings\Administrator\Desktop\pdf>cd dist

C:\Documents and Settings\Administrator\Desktop\pdf\dist>java -jar pdf.jar -s pd
f417 -f JPG -o encoder.jpg "The automated personal identity authentication syste
ms based on iris recognition are reputed to be the most reliable among all biome
tric methods.the iris code false reject rate are comparable to the most accurate
 single finger print matcher.the probabaility of finding two people wit identica
l iris pattern is almost zero"
Barcode4J command-line application, Version DEV

Generating image/jpeg...
done.

C:\Documents and Settings\Administrator\Desktop\pdf\dist>_
```

## OUTPUT OF THE BARCODE IN JPEG FORMAT:

## SNAPSHOT OF DECODER

## EXECUTING THE JAR FILE IN COMMAND PROMPT



C:\WINDOWS\system32\cmd.exe

```
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd Desktop

C:\Documents and Settings\Administrator\Desktop>cd decoder

C:\Documents and Settings\Administrator\Desktop\decoder>cd dist

C:\Documents and Settings\Administrator\Desktop\decoder\dist>java -jar decoder.j
ar encoder.jpg
file:/C:/Documents%20and%20Settings/Administrator/Desktop/decoder/dist/encoder.j
pg (format: PDF417, type: TEXT):
Raw result:
The automated personal identity authentication systems based on iris recognition
 are reputed to be the mosted reliable among all biometric methodsRthe iris code
 false reject rates are comparable to the most accurate single finger fingerprin
t matcherRthe probability of finding the two people with identical iris pattern
is almost zeroDVDKEDAA
Parsed result:
The automated personal identity authentication systems based on iris recognition
 are reputed to be the mosted reliable among all biometric methodsRthe iris code
 false reject rates are comparable to the most accurate single finger fingerprin
t matcherRthe probability of finding the two people with identical iris pattern
is almost zeroDVDKEDAA

C:\Documents and Settings\Administrator\Desktop\decoder\dist>
```

## REFERENCES

1.Implementation of Algorithm to Decode Two-Dimensional Barcode PDF-417, IEEE 26-30 Aug. 2002 page: 1791 - 1794 vol.2

2.TPavlidis, J.Swartz, Y.P. Wang, “Information encoding with two-dimensional barcodes”, IEEE Computer, 1992.

***3.*** Uniform Symbology Specification **PDF417,** AIM **USA.**

**4.**R.C. Gonzalez, R.E.Woods, Digital Image Processing,Addison Wesley, 1993.

5. J.R. Parker, “Gray level thresbolding in badly illuminated images”, IEEE Trans. On PAMI, vol. 13, No. 8, Aug.,**1991.**

**6.** Understanding 2D Symbologies - **a** detailed overview and technical introduction, AIM **USA.**