Java : How to determine the correct charset encoding of a stream

Ask Question

With reference to the following thread: Java App : Unable to read iso-8859-1 encoded file correctly

What is the best way to programatically determine the correct charset encoding of an inputstream/file ?

I have tried using the following:

```
File in =  new File(args[0]);
InputStreamReader r = new InputStreamReader(new FileInputStream(in));
System.out.println(r.getEncoding());
```

But on a file which I know to be encoded with ISO8859_1 the above code yields ASCII, which is not correct, and does not allow me to correctly render the content of the file back to the console.

java file encoding stream character-encoding

edited May 23 '17 at 12:10
Community ♦
1 1

asked Jan 31 '09 at 15:34
Joel
16.4k 29 93 132

11 Eduard is right, "You cannot determine the encoding of a arbitrary byte stream". All other proposals give you ways (and libraries) to do best guessing. But in the end they are still guesses. – Mihai Nita Dec 15 '11 at 12:13

6 `Reader.getEncoding` returns the encoding the reader was set up to use, which in your case is the default encoding. – Karol S Sep 27 '14 at 18:44

14 Answers

I have used this library, similar to jchardet for detecting encoding in Java:
http://code.google.com/p/juniversalchardet/

answered Jan 19 '11 at 13:44
Luciano Fiandesio
7,625 8 38 43

4 I found that this was more accurate: jchardet.sourceforge.net (I was testing on Western European language documents encoded in ISO 8859-1 , windows-1252, utf-8) – Joel Apr 6 '11 at 9:42

This juniversalchardet does not work. It delivers UTF-8 most of time, even if the file is 100% windows-1212 encoded. – Brain Sep 11 '16 at 18:28

You cannot determine the encoding of a arbitrary byte stream. This is the nature of encodings. A encoding means a mapping between a byte value and its representation. So every encoding "could" be the right.

The getEncoding() method will return the encoding which was set up (read the JavaDoc) for the stream. It will not guess the encoding for you.

Some streams tell you which encoding was used to create them: XML, HTML. But not an arbitrary byte stream.

Anyway, you could try to guess an encoding on your own if you have to. Every language has a common frequency for every char. In English the char e appears very often but ê will appear very very seldom. In a ISO-8859-1 stream there are usually no 0x00 chars. But a UTF-16 stream has a lot of them.

Or: you could ask the user. I've already seen applications which present you a snippet of the file in different encodings and ask you to select the "correct" one.

edited Mar 10 '14 at 19:51
Agostino
921 3 20 47

answered Jan 31 '09 at 15:44
Eduard Wirch
7,090 7 49 65

16 This doesn't really answer the question. The op should probably be using docs.codehaus.org/display/GUESSENC/Home or icu-project.org/apiref/icu4j/com/ibm/icu/text/… or jchardet.sourceforge.net – Christoffer Hammarström Dec 15 '10 at 10:23

14 So how does my editor, notepad++ know how to open the file and show me the right characters ? – momomo Dec 20 '11 at 14:51

8 @Hamidam it is by luck that it shows you the right characters. When it guesses wrongly (and it often does), there is an option (Menu >> Encoding) that allows you to change the encoding. – Pacerier Jan 17 '12 at 9:51

10 @Eduard: "So every encoding "could" be the right." not quite right. Many text encodings have several patterns that are invalid, which are a flag that the text is *probably* not that encoding. In fact, given the the first two bytes of a file, only 38% of the combinations are valid UTF8. The odds of the first 5 codepoints being valid UTF8 by chance is less than .77%. Likewise, UTF16BE and LE are usually easily identified by the large number of zero bytes and where they are. – Mooing Duck Dec 6 '12 at 18:32

check this out: http://site.icu-project.org/ (icu4j) they have libraries for detecting charset from IOStream could be simple like this:

```
BufferedInputStream bis = new BufferedInputStream(input);
CharsetDetector cd = new CharsetDetector();
cd.setText(bis);
CharsetMatch cm = cd.detect();

if (cm != null) {
   reader = cm.getReader();
   charset = cm.getName();
}else {
   throw new UnsupportedCharsetException()
}
```

edited Dec 17 '11 at 0:41
Maxim Veksler **10.4k** 26 98 136

answered Oct 25 '10 at 10:11
user345883 **323** 3 5

2 i tried but it greatly fails: i made 2 text files in eclipse both containing "öäüß". One set to iso encoding and one to utf8 - both are detected as utf8! So i tried a file safed somewhere on my hd (windows) - this one was detected correctly ("windows-1252"). Then i created two new file on hd one one edited with editor the other one with notepad++. in both cases "Big5" (Chinese) was detected! – dermoritz Sep 29 '11 at 7:13

2 EDIT: Ok i should check cm.getConfidence() - with my short "äöüß" the confidence is 10. So i have to decide what confidence is good enought - but thats absolutly ok for this endeavour (charset detection) – dermoritz Sep 29 '11 at 7:21

FYI, thank you, this was extremely helpful! – Matthew Hager Mar 7 '14 at 21:23

1 Direct link to sample code: userguide.icu-project.org/conversion/detection – james.garriss Sep 23 '15 at 13:36

Here are my favorites:

TikaEncodingDetector

Dependency:

```
<dependency>
  <groupId>org.apache.any23</groupId>
  <artifactId>apache-any23-encoding</artifactId>
  <version>1.1</version>
</dependency>
```

Sample:

```
public static Charset guessCharset(InputStream is) throws IOException {
  return Charset.forName(new TikaEncodingDetector().guessEncoding(is));
}
```

GuessEncoding

Dependency:

```
<dependency>
  <groupId>org.codehaus.guessencoding</groupId>
  <artifactId>guessencoding</artifactId>
  <version>1.4</version>
  <type>jar</type>
</dependency>
```

Sample:

```
  public static Charset guessCharset2(File file) throws IOException {
    return CharsetToolkit.guessEncoding(file, 4096, StandardCharsets.UTF_8);
  }
```

answered Nov 30 '14 at 12:48
Benny Neugebauer **22.3k** 15 125 130

2 *Nota:* **TikaEncodingDetector 1.1** is actually a thin wrapper around **ICU4J 3.4** `CharsetDectector` class. – Stephan Sep 1 '15 at 17:26

Unfortunately both libs do not work. In one case it identifies a UTF-8 file with german Umlaute as ISO-8859-1 and US-ASCII. – Brain Sep 12 '16 at 11:48

@Brain: Is your tested file actually in an UTF-8 format and does it include a BOM (en.wikipedia.org/wiki/Byte_order_mark)? – Benny Neugebauer Sep 12 '16 at 11:53

@BennyNeugebauer the file is a UTF-8 without BOM. I checked it with Notepad++, also by changing the encoding and asserting that the "Umlaute" still are visible. – Brain Sep 14 '16 at 7:19

You can certainly *validate* the file for a particular charset by decoding it with a `CharsetDecoder` and watching out for "malformed-input" or "unmappable-character" errors. Of course, this only tells you if a charset is wrong; it doesn't tell you if it is correct. For that, you need a basis of comparison to evaluate the decoded results, e.g. do you know beforehand if the characters are restricted to some subset, or whether the text adheres to some strict format? The bottom line is that charset detection is guesswork without any guarantees.

edited Feb 1 '09 at 7:44

answered Feb 1 '09 at 7:33
Zach Scrivena
23.2k 10 53 67

The libs above are simple BOM detectors which of course only work if there is a BOM in the beginning of the file. Take a look at http://jchardet.sourceforge.net/ which does scans the text

answered Feb 15 '10 at 11:53
Lorrat
61 1 1

13 just at tip, but there is no "above" on this site - consider stating the libraries you are referring to. – McDowell Jan 19 '11 at 14:02

I found a nice third party library which can detect actual encoding: http://glaforge.free.fr/wiki/index.php?wiki=GuessEncoding

I didn't test it extensively but it seems to work.

answered Jan 7 '10 at 9:04
falcon
51 1 1

The Link to the "GuessEncoding" project website is: xircles.codehaus.org/p/guessencoding – Benny Neugebauer Oct 26 '14 at 23:16

If you don't know the encoding of your data, it is not so easy to determine, but you could try to use a library to guess it. Also, there is a similar question.

edited May 23 '17 at 12:03
Community ♦
1 1

answered Jan 31 '09 at 15:46
Fabian Steeg
37.4k 5 71 107

For ISO8859_1 files, there is not an easy way to distinguish them from ASCII. For Unicode files however one can generally detect this based on the first few bytes of the file.

UTF-8 and UTF-16 files include a Byte Order Mark (BOM) at the very beginning of the file. The BOM is a zero-width non-breaking space.

Unfortunately, for historical reasons, Java does not detect this automatically. Programs like Notepad will check the BOM and use the appropriate encoding. Using unix or Cygwin, you can check the BOM with the file command. For example:

```
$ file sample2.sql
sample2.sql: Unicode text, UTF-16, big-endian
```

For Java, I suggest you check out this code, which will detect the common file formats and select the correct encoding: How to read a file and automatically specify the correct encoding

edited Dec 10 '11 at 1:03
jdknight
914 17 33

answered May 26 '09 at 7:20
brianegge
19k 11 60 93

11 Not all UTF-8 or UTF-16 files have a BOM, as it is not required, and UTF-8 BOM is discouraged. – Christoffer Hammarström Oct 11 '11 at 13:33

If you use ICU4J (http://icu-project.org/apiref/icu4j/)

Here is my code:

```
String charset = "ISO-8859-1"; //Default chartset, put whatever you want

byte[] fileContent = null;
FileInputStream fin = null;

//create FileInputStream object
fin = new FileInputStream(file.getPath());

/*
 * Create byte array large enough to hold the content of the file.
 * Use File.length to determine size of the file in bytes.
 */
fileContent = new byte[(int) file.length()];
```

```
            /*
             * To read content of the file in byte array, use
             * int read(byte[] byteArray) method of java FileInputStream class.
             *
             */
            fin.read(fileContent);

            byte[] data =  fileContent;

            CharsetDetector detector = new CharsetDetector();
            detector.setText(data);

            CharsetMatch cm = detector.detect();

            if (cm != null) {
                int confidence = cm.getConfidence();
                System.out.println("Encoding: " + cm.getName() + " - Confidence: " +
confidence + "%");
                //Here you have the encode name and the confidence
                //In my case if the confidence is > 50 I return the encode, else I
return the default value
                if (confidence > 50) {
                    charset = cm.getName();
                }
            }
```

Remember to put all the try catch need it.

I hope this works for you.

answered Apr 4 '13 at 21:01
ssamuel68
629 8 8

IMO, this answer is perfectible. If you want to use ICU4j, try this one instead: stackoverflow.com/a/4013565/363573. – Stephan Sep 1 '15 at 15:26

As far as I know, there is no general library in this context to be suitable for all types of problems. So, for each problem you should test the existing libraries and select the best one which satisfies your problem's constraints, but often none of them is appropriate. In these cases you can write your own Encoding Detector! As I have wrote ...

I've wrote a meta java tool for detecting charset encoding of HTML Web pages, using IBM ICU4j and Mozilla JCharDet as the built-in components. Here you can find my tool, please read the README section before anything else. Also, you can find some basic concepts of this problem in my paper and in its references.

Bellow I provided some helpful comments which I've experienced in my work:

- Charset detection is not a foolproof process, because it is essentially based on statistical data and what actually happens is **guessing** not **detecting**
- icu4j is the main tool in this context by IBM, imho
- Both TikaEncodingDetector and Lucene-ICU4j are using icu4j and their accuracy had not a meaningful difference from which the icu4j in my tests (at most %1, as I remember)
- icu4j is much more general than jchardet, icu4j is just a bit biased to IBM family encodings while jchardet is strongly biased to utf-8
- Due to the widespread use of UTF-8 in HTML-world; jchardet is a better choice than icu4j in overall, but is not the best choice!
- icu4j is great for East Asian specific encodings like EUC-KR, EUC-JP, SHIFT_JIS, BIG5 and the GB family encodings
- Both icu4j and jchardet are debacle in dealing with HTML pages with Windows-1251 and Windows-1256 encodings. Windows-1251 aka cp1251 is widely used for Cyrillic-based languages like Russian and Windows-1256 aka cp1256 is widely used for Arabic
- Almost all encoding detection tools are using statistical methods, so the accuracy of output strongly depends on the size and the contents of the input
- Some encodings are essentially the same just with a partial differences, so in some cases the guessed or detected encoding may be false but at the same time be true! As about Windows-1252 and ISO-8859-1. (refer to the last paragraph under the 5.2 section of my paper)

edited Jun 8 '16 at 20:29

answered May 12 '16 at 20:14
faghani
119 1 2 13

Which library to use?

As of this writing, they are three libraries that emerge:

- GuessEncoding
- ICU4j
- juniversalchardet

I don't include Apache Any23 because it uses ICU4j 3.4 under the hood.

How to tell which one has detected the *right* charset (or as close as possible)?

It's impossible to certify the charset detected by each above libraries. However, it's possible to ask them in turn and score the returned response.

How to score the returned response?

Each response can be assigned one point. The more points a response have, the more confidence the detected charset has. This is a simple scoring method. You can elaborate others.

Is there any sample code?

Here is a full snippet implementing the strategy described in the previous lines.

```
public static String guessEncoding(InputStream input) throws IOException {
    // Load input data
    long count = 0;
    int n = 0, EOF = -1;
    byte[] buffer = new byte[4096];
    ByteArrayOutputStream output = new ByteArrayOutputStream();

    while ((EOF != (n = input.read(buffer))) && (count <= Integer.MAX_VALUE)) {
        output.write(buffer, 0, n);
        count += n;
    }

    if (count > Integer.MAX_VALUE) {
        throw new RuntimeException("Inputstream too large.");
    }

    byte[] data = output.toByteArray();

    // Detect encoding
    Map<String, int[]> encodingsScores = new HashMap<>();

    // * GuessEncoding
    updateEncodingsScores(encodingsScores, new
CharsetToolkit(data).guessEncoding().displayName());

    // * ICU4j
    CharsetDetector charsetDetector = new CharsetDetector();
    charsetDetector.setText(data);
    charsetDetector.enableInputFilter(true);
    CharsetMatch cm = charsetDetector.detect();
    if (cm != null) {
        updateEncodingsScores(encodingsScores, cm.getName());
    }

    // * j[illegible]
    Uni[illegible]iversalDetector(null);
    u[illegible]th);
    [illegible]
    String encodingName = universalDetector.getDetectedCharset();
    if (encodingName != null) {
        updateEncodingsScores(encodingsScores, encodingName);
    }

    // Find winning encoding
    Map.Entry<String, int[]> maxEntry = null;
    for (Map.Entry<String, int[]> e : encodingsScores.entrySet()) {
        if (maxEntry == null || (e.getValue()[0] > maxEntry.getValue()[0])) {
            maxEntry = e;
        }
    }

    String winningEncoding = maxEntry.getKey();
    //dumpEncodingsScores(encodingsScores);
    return winningEncoding;
}

private static void updateEncodingsScores(Map<String, int[]> encodingsScores, String
encoding) {
    String encodingName = encoding.toLowerCase();
    int[] encodingScore = encodingsScores.get(encodingName);

    if (encodingScore == null) {
        encodingsScores.put(encodingName, new int[] { 1 });
    } else {
        encodingScore[0]++;
    }
}

private static void dumpEncodingsScores(Map<String, int[]> encodingsScores) {
    System.out.println(toString(encodingsScores));
}

private static String toString(Map<String, int[]> encodingsScores) {
    String GLUE = ", ";
    StringBuilder sb = new StringBuilder();

    for (Map.Entry<String, int[]> e : encodingsScores.entrySet()) {
        sb.append(e.getKey() + ":" + e.getValue()[0] + GLUE);
    }
    int len = sb.length();
    sb.delete(len - GLUE.length(), len);

    return "{ " + sb.toString() + " }";
}
```

Improvements: The `guessEncoding` method reads the inputstream entirely. For large inputstreams this can be a concern. All these libraries would read the whole inputstream. This would imply a large time consumption for detecting the charset.

It's possible to limit the initial data loading to a few bytes and perform the charset detection on those few bytes only.

edited Jul 31 '17 at 7:58
DNA
33.1k 10 71 113

answered Sep 3 '15 at 10:38
Stephan
25.2k 28 141 239

An alternative to TikaEncodingDetector is to use Tika AutoDetectReader.

```
Charset charset = new AutoDetectReader(new FileInputStream(file)).getCharset();
```

edited Sep 3 '15 at 9:47
Stephan
25.2k 28 141 239

answered May 11 '15 at 13:04
Nolf
93 1 7

Tike AutoDetectReader uses EncodingDetector loaded with ServiceLoader. Which EncodingDetector implementations do you use? – Stephan Sep 3 '15 at 9:48

Can you pick the appropriate char set in the Constructor:

```
new InputStreamReader(new FileInputStream(in), "ISO8859_1");
```

answered Jan 31 '09 at 15:44
Kevin
26.1k 8 65 72

6 The point here was to see whether the charset could be determined programatically. – Joel Jan 31 '09 at 15:46

1 No, it won't guess it for you. You have to supply it. – Kevin Jan 31 '09 at 15:50

1 There may be a heuristic method, as suggested by some of the answers here stackoverflow.com/questions/457655/java-charset-and-windows/… – Joel Jan 31 '09 at 15:56