Charset Normalizer Documentation

Release 1.3.0

Ahmed TAHRI

Aug 21, 2023

Contents

CHAPTER 1

Overview

Library that help you read text from unknown charset encoding. Project motivated by chardet, I'm trying to resolve the issue by taking another approach. All IANA character set names for which the Python core library provides codecs are supported.

CHAPTER 2

Introduction

This library aim to assist you in finding what encoding suit the best to content. It **DOES NOT** try to uncover the originating encoding, in fact this program does not care about it.

By originating we means the one that was precisely used to encode a text file.

Precisely

```
my_byte_str = 'Bonjour, je suis à la recherche d\'une aide sur les étoiles'.encode(
↪'cp1252')
```

We **ARE NOT** looking for cp1252 **BUT FOR** `Bonjour, je suis à la recherche d'une aide sur les étoiles`. Because of this

```
my_byte_str.decode('cp1252') == my_byte_str.decode('cp1256') == my_byte_str.decode(
↪'cp1258') == my_byte_str.decode('iso8859_14')
# Print True !
```

There is no wrong answer to decode `my_byte_str` to get the exact same result. This is where this library differ from others. There's not specific probe per encoding table.

CHAPTER 3

Features

- Encoding detection on a stream, bytes or file.
- Transpose any encoded content to Unicode the best we can.
- Detect spoken language in text.

Contents:

3.1 Support

Here are a list of supported encoding and supported language with latest update. Also this list may change depending of your python version.

3.1.1 Supported Encodings

Charset Normalizer is able to detect any of those encoding.

IANA Code Page	Aliases
ascii	646, ansi_x3.4_1968, ansi_x3_4_1968, ansi_x3.4_1986, cp367, csascii, ibm367, iso646_us, iso_646.irv_1991, i
big5	big5_tw, csbig5, x_mac_trad_chinese
big5hkscs	big5_hkscs, hkscs
cp037	037, csibm037, ebcdic_cp_ca, ebcdic_cp_nl, ebcdic_cp_us, ebcdic_cp_wt, ibm037, ibm039
cp1026	1026, csibm1026, ibm1026
cp1125	1125, ibm1125, cp866u, ruscii
cp1140	1140, ibm1140
cp1250	1250, windows_1250
cp1251	1251, windows_1251
cp1252	1252, windows_1252
cp1253	1253, windows_1253

Conti

Table 1 – continued from previous page

IANA Code Page	Aliases
cp1254	1254, windows_1254
cp1255	1255, windows_1255
cp1256	1256, windows_1256
cp1257	1257, windows_1257
cp1258	1258, windows_1258
cp273	273, ibm273, csibm273
cp424	424, csibm424, ebcdic_cp_he, ibm424
cp437	437, cspc8codepage437, ibm437
cp500	500, csibm500, ebcdic_cp_be, ebcdic_cp_ch, ibm500
cp775	775, cspc775baltic, ibm775
cp850	850, cspc850multilingual, ibm850
cp852	852, cspcp852, ibm852
cp855	855, csibm855, ibm855
cp857	857, csibm857, ibm857
cp858	858, csibm858, ibm858
cp860	860, csibm860, ibm860
cp861	861, cp_is, csibm861, ibm861
cp862	862, cspc862latinhebrew, ibm862
cp863	863, csibm863, ibm863
cp864	864, csibm864, ibm864
cp865	865, csibm865, ibm865
cp866	866, csibm866, ibm866
cp869	869, cp_gr, csibm869, ibm869
cp932	932, ms932, mskanji, ms_kanji
cp949	949, ms949, uhc
cp950	950, ms950
euc_jis_2004	jisx0213, eucjis2004, euc_jis2004
euc_jisx0213	eucjisx0213
euc_jp	eucjp, ujis, u_jis
euc_kr	euckr, korean, ksc5601, ks_c_5601, ks_c_5601_1987, ksx1001, ks_x_1001, x_mac_korean
gb18030	gb18030_2000
gb2312	chinese, csiso58gb231280, euc_cn, euccn, eucgb2312_cn, gb2312_1980, gb2312_80, iso_ir_58, x_mac_simp_c
gbk	936, cp936, ms936
hp_roman8	roman8, r8, csHPRoman8
hz	hzgb, hz_gb, hz_gb_2312
iso2022_jp	csiso2022jp, iso2022jp, iso_2022_jp
iso2022_jp_1	iso2022jp_1, iso_2022_jp_1
iso2022_jp_2	iso2022jp_2, iso_2022_jp_2
iso2022_jp_3	iso2022jp_3, iso_2022_jp_3
iso2022_jp_ext	iso2022jp_ext, iso_2022_jp_ext
iso2022_kr	csiso2022kr, iso2022kr, iso_2022_kr
iso8859_10	csisolatin6, iso_8859_10, iso_8859_10_1992, iso_ir_157, l6, latin6
iso8859_11	thai, iso_8859_11, iso_8859_11_2001
iso8859_13	iso_8859_13, l7, latin7
iso8859_14	iso_8859_14, iso_8859_14_1998, iso_celtic, iso_ir_199, l8, latin8
iso8859_15	iso_8859_15, l9, latin9
iso8859_16	iso_8859_16, iso_8859_16_2001, iso_ir_226, l10, latin10
iso8859_2	csisolatin2, iso_8859_2, iso_8859_2_1987, iso_ir_101, l2, latin2
iso8859_3	csisolatin3, iso_8859_3, iso_8859_3_1988, iso_ir_109, l3, latin3

Conti

Table 1 – continued from previous page

IANA Code Page	Aliases
iso8859_4	csisolatin4, iso_8859_4, iso_8859_4_1988, iso_ir_110, l4, latin4
iso8859_5	csisolatincyrillic, cyrillic, iso_8859_5, iso_8859_5_1988, iso_ir_144
iso8859_6	arabic, asmo_708, csisolatinarabic, ecma_114, iso_8859_6, iso_8859_6_1987, iso_ir_127
iso8859_7	csisolatingreek, ecma_118, elot_928, greek, greek8, iso_8859_7, iso_8859_7_1987, iso_ir_126
iso8859_8	csisolatinhebrew, hebrew, iso_8859_8, iso_8859_8_1988, iso_ir_138
iso8859_9	csisolatin5, iso_8859_9, iso_8859_9_1989, iso_ir_148, l5, latin5
iso2022_jp_2004	iso_2022_jp_2004, iso2022jp_2004
johab	cp1361, ms1361
koi8_r	cskoi8r
kz1048	kz_1048, rk1048, strk1048_2002
latin_1	8859, cp819, csisolatin1, ibm819, iso8859, iso8859_1, iso_8859_1, iso_8859_1_1987, iso_ir_100, l1, latin, latin
mac_cyrillic	maccyrillic
mac_greek	macgreek
mac_iceland	maciceland
mac_latin2	maccentraleurope, maclatin2
mac_roman	macintosh, macroman
mac_turkish	macturkish
mbcs	ansi, dbcs
ptcp154	csptcp154, pt154, cp154, cyrillic_asian
rot_13	rot13
shift_jis	csshiftjis, shiftjis, sjis, s_jis, x_mac_japanese
shift_jis_2004	shiftjis2004, sjis_2004, s_jis_2004
shift_jisx0213	shiftjisx0213, sjisx0213, s_jisx0213
tactis	tis260
tis_620	tis620, tis_620_0, tis_620_2529_0, tis_620_2529_1, iso_ir_166
utf_16	u16, utf16
utf_16_be	unicodebigunmarked, utf_16be
utf_16_le	unicodelittleunmarked, utf_16le
utf_32	u32, utf32
utf_32_be	utf_32be
utf_32_le	utf_32le
utf_8	u8, utf, utf8, utf8_ucs2, utf8_ucs4

3.1.2 Supported Languages

Those language can be detected inside your content. All of these are specified in ./charset_normalizer/assets/frequencies.json .

English, German, French, Dutch, Italian, Polish, Spanish, Russian, Japanese, Portuguese, Swedish, Chinese, Catalan, Ukrainian, Norwegian, Finnish, Vietnamese, Czech, Hungarian, Korean, Indonesian, Turkish, Romanian, Farsi, Arabic, Danish, Esperanto, Serbian, Lithuanian, Slovene, Slovak, Malay, Hebrew, Bulgarian, Kazakh, Baque, Volapük, Croatian, Hindi, Estonian, Azeri, Galician, Simple English, Nynorsk, Thai, Greek, Macedonian, Serbocroatian, Tamil, Classical Chinese.

3.2 Installation

This installs a package that can be used from Python (`import charset_normalizer`).

To install for all users on the system, administrator rights (root) may be required.

3.2.1 Using PIP

Charset Normalizer can be installed from pip:

```
pip install charset-normalizer
```

You may enable extra feature unicode data v12 backport as follow:

```
pip install charset-normalizer[UnicodeDataBackport]
```

3.2.2 From git via master

You can install from dev-master branch using git:

```
git clone https://github.com/Ousret/charset_normalizer.git
cd charset_normalizer/
python setup.py install
```

3.3 Basic Usage

3.3.1 The new way

You may want to get right to it.

```
from charset_normalizer import CharsetNormalizerMatches as CnM

# This is going to print out your sequence once properly decoded
print(
    CnM.from_bytes(
        my_byte_str
    ).best().first()
)

# You could also want the same from a file
print(
    CnM.from_path(
        './data/sample.1.ar.srt'
    ).best().first()
)
```

3.3.2 Backward compatibility

If you were used to python chardet, we are providing the very same `detect()` method as chardet.

```
from charset_normalizer import detect

# This will behave exactly the same as python chardet
result = detect(my_byte_str)

if result['encoding'] is not None:
    print('got', result['encoding'], 'as detected encoding')
```

You may upgrade your code with ease. CTRL + R `from chardet import detect` to `from charset_normalizer import detect`.

3.4 Advanced Search

Charset Normalizer method `from_bytes`, `from_fp` and `from_path` provide some optional parameters that can be tweaked.

As follow

```
from charset_normalizer import CharsetNormalizerMatches as CnM

my_byte_str = ''.encode('gb18030')

results = CnM.from_bytes(
    my_byte_str,
    steps=10,  # Number of steps/block to extract from my_byte_str
    chunk_size=512,  # Set block size of each extraction
    threshold=0.2,  # Maximum amount of chaos allowed on first pass
    cp_isolation=None,  # Finite list of encoding to use when searching for a match
    cp_exclusion=None,  # Finite list of encoding to avoid when searching for a match
    preemptive_behaviour=True,  # Determine if we should look into my_byte_str (ASCII-
↪Mode) for pre-defined encoding
    explain=False  # Print on screen what is happening when searching for a match
)
```

3.4.1 Using CharsetNormalizerMatches

Here, `results` is a `CharsetNormalizerMatches` object. It behave like a list. Initially it is not sorted. Be cautious when extracting `first()` result without calling method `best()`.

3.4.2 List behaviour

Like said earlier, `CharsetNormalizerMatches` object behave like a list.

```
# Call len on results also work
if len(results) == 0:
    print('No match for your sequence')

# Iterate over results like a list
for match in results:
    print(match.encoding, 'can decode properly your sequence using', match.
↪alphabets, 'and language', match.language)

# Using index to access results
if len(results) > 0:
    print(str(results[0]))
```

3.4.3 Using best()

Like said above, `CharsetNormalizerMatches` object behave like a list and it is not sorted after calling `from_bytes`, `from_fp` or `from_path`.

Using `best()` keep only the lowest chaotic results and in it the best coherent result if necessary. It produce also a `CharsetNormalizerMatches` object as return value.

```
results = results.best()
```

3.4.4 Calling first()

This method is callable from a `CharsetNormalizerMatches` object. It extract the first match in list. This method return a `CharsetNormalizerMatch` object. See Handling result section.

3.4.5 Class aliases

`CharsetNormalizerMatches` is also known as `CharsetDetector`, `CharsetDoctor` and `EncodingDetector`. It is useful if you prefer short class name.

3.4.6 Verbose output

You may want to understand why a specific encoding was not picked by charset_normalizer. All you have to do is passing `explain` to True when using methods `from_bytes`, `from_fp` or `from_path`.

3.5 Handling Result

When initiating search upon a buffer, bytes or file you can assign the return value and fully exploit it.

```
my_byte_str = ''.encode('gb18030')

# Assign return value so we can fully exploit result
result = CnM.from_bytes(
    my_byte_str
).best().first()

print(result.encoding)  # gb18030
```

3.5.1 Using CharsetNormalizerMatch

Here, `result` is a `CharsetNormalizerMatch` object or `None`.

3.6 Miscellaneous

3.6.1 Convert to str

Any `CharsetNormalizerMatch` object can be transformed to exploitable `str` variable.

```
my_byte_str = ''.encode('gb18030')

# Assign return value so we can fully exploit result
result = CnM.from_bytes(
```

(continues on next page)

(continued from previous page)

```
    my_byte_str
).best().first()

# This should print ''
print(str(result))
```

3.6.2 Expect UnicodeDecodeError

This package also offer you the possibility to reconfigure the way `UnicodeDecodeError` is raised. Charset Normalizer offer the possibility to extend the actual message inside it to provide a clue about what encoding it should actually be.

```
import charset_normalizer  # Nothing else is needed

my_byte_str = ''.encode('gb18030')
my_byte_str.decode('utf_8')  # raise UnicodeDecodeError

# UnicodeDecodeError: 'utf-8' codec can't decode byte 0xce in position 0:␣
↪invalid continuation byte; you may want to consider gb18030 codec for this␣
↪sequence.
# instead of
# UnicodeDecodeError: 'utf-8' codec can't decode byte 0xce in position 0:␣
↪invalid continuation byte
```

Here, the addition is “you may want to consider gb18030 codec for this sequence.”. Is does not work when using `try .. except` block.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search