baudot Documentation

Release 0.1.1.post2

Author

Mar 17, 2019

Contents:

Baudot is a Python library for encoding and decoding 5-bit stateful encoding.

This library is named after Jean-Maurice-Émile Baudot (1845-1903), the French engineer who invented this code. The Baudot code was the first practical and widely used binary character encoding, and is an ancestor of the ASCII code we are familiar with today.

CHAPTER 1

About this library and the Baudot code

1.1 What is the Baudot code?

The Baudot code was the first (or at least the first practical) fixed-length character encoding to be used widely in the telecommunications industry. This system, invented and patented by the French engineer Jean-Maurice-Émile Baudot in 1870, was intended as a replacement for Morse code when sending telegraph messages. It allowed the use of a machine (also patented) to read the messages automatically.

However the code still had to be composed manually; in 1901 the process was refined by the American engineer Donald Murray, so that it could be easily composed on a typewriter-like machine. The code was also modified to reduce and optimize the wear on the tape-punching mechanism. This system, known as the Baudot-Murray code or ITA2, was even more widely adopted and vastly used through World War II.

This new standard was eventually one of the bases for the design of the ASCII encoding that we are now familiar with. In retrospective, the legacy of the Baudot and Murray codes is immense, though they are rarely used today.

1.2 How did it work?

The Baudot code (and Baudot-Murray afterward) is a 5-bit stateful binary code. This is a modern description though, since at the time these "bits" would have just been holes in paper tapes.

Because each line of tape can hold five holes/bits, that means that the code allows 32 possible combinations per character. This however is obviously not enough to hold the 26 letters of the alphabet plus ten digits, let alone other symbols. Baudot's solution was to use special "shift" characters, which would indicate whether the following codes (until the next shift) were letters or numbers (and symbols). Hence why it is called a "stateful" encoding. This is unlike ASCII and its successors, where each character has its unique code.

The Baudot-Murray code extends on the idea of control characters, introducing codes such as "Carriage Return", "Line Feed", "Enquiry" and "Bell". There even exists a variant of ITA2 for Russian use, which introduces a third shift that exposes a table of cyrillic characters.

1.3 So, why this library?

I got interested in 5-bit encoding while learning about the now famous code breaking efforts lead by the United Kingdom during WWII. Such tapes were even an essential component of Colossus, the first electronic computer which was designed for decrypting the German Lorenz cipher.

At first I thought decoding this could be a fun exercise, then discovered that I could not find any Python library on PyPi for doing that. So here I am, doing this for fun (and so that I could call dibs on the "baudot" name).

Quite honestly, I cannot think of many good use cases for this library. Reportedly, ITA2 is still commonly used in the radio amateur community, so that could be a potential one. Or this could be used to make a simulation of the Colossus computer.

1.4 More resources

- Baudot Code - Wikipedia
- 5 Hole Paper Tape - Computerphile

CHAPTER 2

User Guide

2.1 Library walk-through

`baudot.core` This module holds the stateful encoding/decoding logic. Its functions are directly available in `baudot` for convenience.

`baudot.codecs` This package hosts the lookup tables, used for encoding/decoding single characters. Standard ITA1 and ITA2 tables are built-in, and the tools for making custom codes are also provided.

`baudot.handlers` This package provides writer and reader classes for a variety of input and output formats.

`baudot.exceptions` As its name suggests, this module defines the library's exceptions. All are subclasses of `BaudotException`.

2.2 Basic usage

The core functions for any operation in this library are `baudot.encode()` and `baudot.decode()`.

To work, both require three elements:

1. a text input (for encoding) or output (for decoding) stream
2. a codec object
3. a reader (for decoding) or writer (for encoding) object

This is because overall, `baudot` accomplishes two tasks (and their inverse):

1. reading 5-bit codes from custom input formats,
2. converting 5-bit codes to unicode characters.

Codec objects are instances of `baudot.codecs.BaudotCodec` (or its sub-classes, to be more specific). A codec is a static object capable of converting characters to codes and back. This library includes a few default codecs but others may be user-defined.

Readers and writers are instances of *baudot.handlers.BaudotReader* and *baudot.handlers.BaudotWriter* respectively. Currently, all the handlers in this library require a stream to be passed at instantiation, that they will read from or write to. This mimics the way it's done in the standard library module csv.

The reason I/O in this library depends on streams is so that many types of inputs and outputs are natively supported, such as files or `stdin` and `stdout`. Or maybe odd devices that natively support Baudot code! This however can be inconvenient for small tests, so two helper functions *baudot.encode_str()* and *baudot.decode_to_str()* are available for using strings as text input. Maybe the handlers could be fitted with a similar feature in the future.

Please keep in mind that this project is very young, and that its API is most likely ill-designed at this point. Suggestions are welcome!

2.3 Examples

2.3.1 Encoding example

```
from io import StringIO
from baudot import encode_str, codecs, handlers

input_str = 'HELLO WORLD!'
with StringIO() as output_buffer:
    writer = handlers.TapeWriter(output_buffer)
    encode_str(input_str, codecs.ITA2_STANDARD, writer)
    print(output_buffer.getvalue())
```

This would output the following:

```
***.**
* *.
   . *
*  .*
*  .*
** .
  *.
*  .**
** .
 * .*
*  .*
 * . *
** .**
 **. *
```

2.3.2 Decoding example

```
from io import BytesIO
from baudot import decode_to_str, codecs, handlers

code = b'1f14011212180413180a12091b0d'
with BytesIO(code) as code_stream:
    reader = handlers.HexBytesReader(code_stream)
    print(decode_to_str(reader, codecs.ITA2_US))
```

Should print:

```
HELLO WORLD!
```

CHAPTER 3

API Reference

3.1 baudot

Baudot – Tools for handling stateful 5-bit encoding

baudot.**decode**(*reader: baudot.handlers.core.BaudotReader*, *codec: baudot.codecs.core.BaudotCodec*, *stream: io.TextIOBase*)

Decode a baudot code stream from a reader to a unicode stream, using a given codec.

Parameters

- **reader** – Reader instance that will read codes from an input
- **codec** – Codec to use for decoding
- **stream** – Unicode stream to write to (can be a file)

baudot.**decode_to_str**(*reader: baudot.handlers.core.BaudotReader*, *codec: baudot.codecs.core.BaudotCodec*) → str

Decode a baudot code stream from a reader to a unicode string, using a given codec.

Parameters

- **reader** – Reader instance that will read codes from an input
- **codec** – Codec to use for decoding

Returns Decoded Unicode string

baudot.**encode**(*stream: io.TextIOBase*, *codec: baudot.codecs.core.BaudotCodec*, *writer: baudot.handlers.core.BaudotWriter*)
Encode unicode characters from an input stream to an output writer, using the given codec.

Parameters

- **stream** – Unicode character stream to encode (can be a file)
- **codec** – Codec to use for encoding
- **writer** – Writer instance for the wanted output format

baudot.**encode_str**(*chars: str*, *codec: baudot.codecs.core.BaudotCodec*, *writer: baudot.handlers.core.BaudotWriter*)
Encode unicode characters from an input string to an output writer, using the given codec.

Parameters

- **chars** – Unicode string to encode
- **codec** – Codec to use for encoding
- **writer** – Writer instance for the wanted output format

3.2 baudot.core

Core encoding/decoding logic of the library

All tools from this module are available from *baudot* for convenience.

3.3 baudot.codecs

Codecs are the tools used to convert encoded-data (5-bit digits) into Unicode characters and back.

baudot.codecs.**ITA1_CONTINENTAL**
Codec for the original Baudot code, a.k.a. ITA1 continental

baudot.codecs.**ITA2_STANDARD**
Codec for the standard Baudot-Murray code, a.k.a. ITA2

baudot.codecs.**ITA2_US**
Codec for the US variant of the Baudot-Murray code, a.k.a. US-TTY

class baudot.codecs.**BaudotCodec**
Bases: abc.ABC

Abstract Base Class for a Codec

Subclasses must implement *encode()* and *decode()*

decode(*code: int*, *state: baudot.codecs.core.Shift*) → Union[str, baudot.codecs.core.Shift]
Abstract method for decoding a single code.

encode(*value: Union[str, baudot.codecs.core.Shift]*, *state: baudot.codecs.core.Shift*) → Tuple[int, baudot.codecs.core.Shift]
Abstract method for encoding a single character or state shift

class baudot.codecs.**Shift**(*name*)
Bases: tuple

name
Alias for field number 0

class baudot.codecs.**SimpleTabledCodec**(*name: str, tables: Dict[baudot.codecs.core.Shift, List[Union[str, baudot.codecs.core.Shift]]]*)
Bases: baudot.codecs.core.BaudotCodec

Creates a codec based on a character table.

The input format must be a dictionary of which the keys are the possible states (instances of Shift) and the values are lists of length 32 exactly, containing characters or shifts.

The Shift instances are the only control characters this library knows of. Any other must be taken from ASCII/Unicode.

decode(*code: int, state: Optional[baudot.codecs.core.Shift]*) → Union[str, baudot.codecs.core.Shift]
Get the character or state shift corresponding to a given code in a given state.

Parameters

- **code** – Code to look up
- **state** – State to apply. This may be *None*, so that a the state can be initialized.

Returns Decoded character or state shift

encode(*value: Union[str, baudot.codecs.core.Shift], state: baudot.codecs.core.Shift*) → Tuple[int, baudot.codecs.core.Shift]
Get the code of the given character of Shift for this codec.

Actually, this logic returns not only the code but also the state required for this code. The current state should also be passed so that more complicated cases can be solved.

Parameters

- **value** – Value (character or state shift) to encode
- **state** – Current state of encoding

Returns Code for this value, and required state

3.4 baudot.handlers

The handlers are interfaces to read and write 5-bit data from a variety of formats.

class baudot.handlers.**BaudotReader**
Bases: abc.ABC

Abstract Base Class for a reader

class baudot.handlers.**BaudotWriter**
Bases: abc.ABC

Abstract Base Class for a writer

write(*code: int*)
Write a single code to the output

3.4.1 baudot.handlers.hexbytes

Handler for reading and writing 5-bit codes as a hexadecimal bit stream.

class baudot.handlers.hexbytes.**HexBytesReader**(*stream: io.BufferedIOBase*)
Bases: baudot.handlers.core.BaudotReader

Reader for hexadecimal 5-bit streams

class baudot.handlers.hexbytes.**HexBytesWriter**(*stream: io.BufferedIOBase*)
Bases: baudot.handlers.core.BaudotWriter

Writer for hexadecimal 5-bit stream

write(*code: int*)
Writes a code as an hexadecimal value

3.4.2 baudot.handlers.tape

Handler for reading and writing to pretty tape-like formatted text

For example, the tape might look like::

```
***.**
* *.
   . *
*  .*
*  .*
** .
  *.
*  .**
** .
 * .*
*  .*
 * . *
** .**
 **. *
```

(Which reads 'HELLO WORLD!')

class baudot.handlers.tape.**TapeConfig**
Bases: tuple

Object for storing a tape representation format.

blank
Alias for field number 1

punch
Alias for field number 0

sep
Alias for field number 2

class baudot.handlers.tape.**TapeReader**(*stream: io.TextIOBase*, *config: baudot.handlers.tape.TapeConfig = TapeConfig(punch='*', blank=' ', sep='.')*)
Bases: baudot.handlers.core.BaudotReader

Reader class for tape-like data.

class baudot.handlers.tape.**TapeWriter**(*stream: io.TextIOBase*, *config: baudot.handlers.tape.TapeConfig = TapeConfig(punch='*', blank=' ', sep='.')*)

Bases: baudot.handlers.core.BaudotWriter

Writer class for tape-like data.

write(*code: int*)

Writes a code to tape

3.5 baudot.exceptions

Custom exceptions for the Baudot library

exception baudot.exceptions.**BaudotException**

Bases: Exception

Core exception class for this library

exception baudot.exceptions.**DecodingError**

Bases: *baudot.exceptions.BaudotException*

Raised on decoding error

exception baudot.exceptions.**EncodingError**

Bases: *baudot.exceptions.BaudotException*

Raised on encoding error

exception baudot.exceptions.**IncoherentTable**

Bases: *baudot.exceptions.BaudotException*

Raised when an encoding/decoding table is not valid

exception baudot.exceptions.**ReadError**

Bases: *baudot.exceptions.BaudotException*

Raised when reading a 5-bit stream fails

exception baudot.exceptions.**WriteError**

Bases: *baudot.exceptions.BaudotException*

Raised when writing a 5-bit stream fails

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index