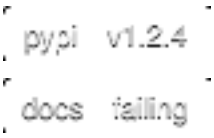
Graph Transliterator Documentation

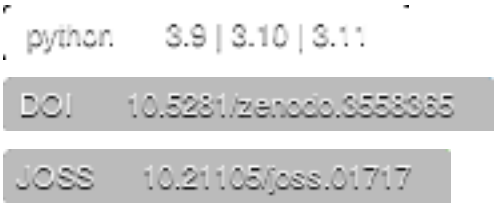
Release 1.2.4

A. Sean Pue

Oct 16, 2023

CONTENTS





A graph-based transliteration tool that lets you convert the symbols of one language or script to those of another using rules that you define.

- Free software: MIT license
- Documentation: https://graphtransliterator.readthedocs.io
- Repository: https://github.com/seanpue/graphtransliterator

CHAPTER

ONE

TRANSLITERATION... WHAT? WHY?

Moving text or data from one script or encoding to another is a common problem:

- Many languages are written in multiple scripts, and many people can only read one of them. Moving between them can be a complex but necessary task in order to make texts accessible.
- The identification of names and locations, as well as machine translation, benefit from transliteration.
- Library systems often require metadata be in particular forms of romanization in addition to the original script.
- Linguists need to move between different methods of phonetic transcription.
- Documents in legacy fonts must now be converted to contemporary Unicode ones.
- Complex-script languages are frequently approached in natural language processing and in digital humanities research through transliteration, as it provides disambiguating information about pronunciation, morphological boundaries, and unwritten elements not present in the original script.

Graph Transliterator abstracts transliteration, offering an "easy reading" method for developing transliterators that does not require writing a complex program. It also contains bundled transliterators that are rigorously tested. These can be expanded to handle many transliteration tasks.

Contributions are very welcome!

CHAPTER

TWO

FEATURES

- Provides a transliteration tool that can be configured to convert the tokens of an input string into an output string using:
  - user-defined types of input **tokens** and **token classes**
  - **transliteration rules** based on:
    - a sequence of input tokens
    - specific input tokens that precede or follow the token sequence
    - classes of input tokens preceding or following specified tokens
  - **"on match" rules** for output to be inserted between transliteration rules involving particular token classes
  - defined rules for **whitespace**, including its optional consolidation
- Can be setup using:
  - an **"easy reading"** YAML format that lets you quickly craft settings for the transliteration tool
  - a JSON dump of a transliterator (quicker!)
  - **"direct"** settings, perhaps passed programmatically, using a dictionary
- **Automatically orders rules** by the number of tokens in a transliteration rule
- **Checks for ambiguity** in transliteration rules
- Can provide **details** about each transliteration rule match
- Allows **optional matching of all possible rules** in a particular location
- Permits **pruning of rules** with certain productions
- **Validates**, as well as **serializes** to and **deserializes** from JSON and Python data types, using accessible marshmallow schemas
- Provides **full support for Unicode**, including Unicode **character names** in the "easy reading" YAML format
- Constructs and uses a **directed tree** and performs a **best-first search** to find the most specific transliteration rule in a given context
- Includes **bundled transliterators** that *you* can add to hat check for full test coverage of the nodes and edges of the internal graph and any "on match" rules
- Includes a command-line interface to perform transliteration and other tasks

CHAPTER

THREE

SAMPLE CODE AND GRAPH

```
from graphtransliterator import GraphTransliterator
GraphTransliterator.from_yaml("""
    tokens:
      h: [consonant]
      i: [vowel]
      " ": [whitespace]
    rules:
      h: \N{LATIN SMALL LETTER TURNED I}
      i: \N{LATIN SMALL LETTER TURNED H}
      <whitespace> i: \N{LATIN CAPITAL LETTER TURNED H}
      (<whitespace> h) i: \N{LATIN SMALL LETTER TURNED H}!
    onmatch_rules:
      - <whitespace> + <consonant>: ¡
    whitespace:
      default: " "
      consolidate: true
      token_class: whitespace
    metadata:
      title: "Upside Down Greeting Transliterator"
      version: "1.0.0"
""").transliterate("hi")
```

```
'¡ʜʏ!'
```

3.1 Get It Now

```
$ pip install -U graphtransliterator
```

3.2 Citation

To cite Graph Transliterator, please use:

> Pue, A. Sean (2019). Graph Transliterator: A graph-based transliteration tool. Journal of Open Source Software, 4(44), 1717, https://doi.org/10.21105/joss.01717

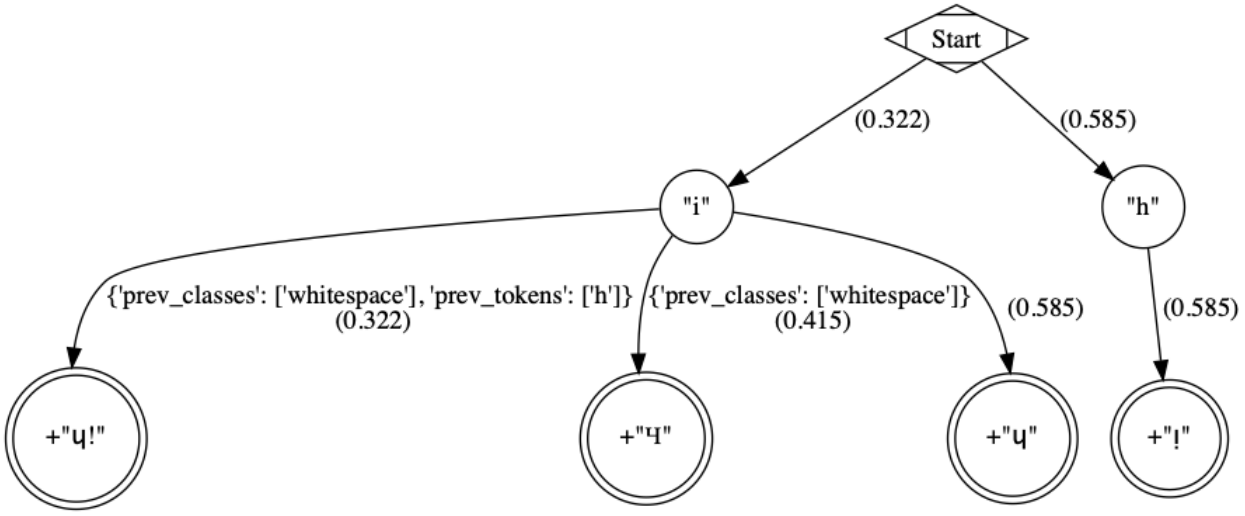


Fig. 1: Sample directed tree created by Graph Transliterator. The *rule* nodes are in double circles, and *token* nodes are single circles. The numbers are the cost of the particular edge, and less costly edges are searched first. Previous token classes and previous tokens that must be present are found as constraints on the edges incident to the terminal leaf *rule* nodes.

3.2.1 Installation

Stable release

To install Graph Transliterator, run this command in your terminal:

```
$ pip install graphtransliterator
```

This is the preferred method to install Graph Transliterator, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Graph Transliterator can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/seanpue/graphtransliterator
```

Or download the tarball:

```
$ curl  -OL https://github.com/seanpue/graphtransliterator/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Required modules

Graph Transliterator requires three Python modules, click, marshmallow and pyyaml. These modules will be installed automatically using the methods described above.

3.2.2 Usage

To use Graph Transliterator in a project:

```
from graphtransliterator import GraphTransliterator
```

Overview

Graph Transliterator requires that you first configure a `GraphTransliterator`. Then you can transliterate an input string using `transliterate()`. There are a few additional methods that can be used to extract information for specific use cases, such as details about which rules were matched.

Configuration

Graph Transliterator takes the following parameters:

1. The acceptable types of **tokens** in the input string as well as any associated **token classes**.
2. The **transliteration rules** for the transformation of the input string.
3. Rules for dealing with **whitespace**.
4. **"On match" rules** for strings to be inserted in particular contexts right before a transliteration rule's output is added (optional).
5. **Metadata** settings for the transliterator (optional).

Initialization

Defining the rules for transliteration can be difficult, especially when dealing with complex scripts. That is why Graph Transliterator uses an "easy reading" format that allows you to enter the transliteration rules in the popular YAML format, either from a string (using `from_yaml()`) or by reading from a file or stream (`GraphTransliterator.from_yaml_file()`). You can also initialize from the loaded contents of YAML (`GraphTransliterator.from_easyreading_dict()`).

Here is a quick sample that parameterizes `GraphTransliterator` using an easy reading YAML string (with comments):

```
yaml_ = """
  tokens:
    a: [vowel]               # type of token ("a") and its class (vowel)
    bb: [consonant, b_class] # type of token ("bb") and its classes (consonant, b_
↪class)
    ' ': [wb]                # type of token (" ") and its class ("wb", for wordbreak)
  rules:
    a: A       # transliterate "a" to "A"
    bb: B      # transliterate "bb" to "B"
    a a: <2AS> # transliterate ("a", "a") to "<2AS>"
```

(continues on next page)

(continued from previous page)

```
    ' ': ' '   # transliterate ' ' to ' '
  whitespace:
    default: " "        # default whitespace token
    consolidate: false  # whitespace should not be consolidated
    token_class: wb     # whitespace token class
"""
gt_one = GraphTransliterator.from_yaml(yaml_)
gt_one.transliterate('a')
```

```
'A'
```

```
gt_one.transliterate('bb')
```

```
'B'
```

```
gt_one.transliterate('aabb')
```

```
'<2AS>B'
```

The example above shows a very simple transliterator that replaces the input token “a” with “A”, “bb” with “B”, ” ” with ” “, and two “a” in a row with “<2AS>”. It does not consolidate whitespace, and treats ” ” as its default whitespace token. Tokens contain strings of one or more characters.

Input Tokens and Token Class Settings

During transliteration, Graph Transliterator first attempts to convert the input string into a list of tokens. This is done internally using `GraphTransliterator.tokenize()`:

```
gt_one.tokenize('abba')
```

```
[' ', 'a', 'bb', 'a', ' ']
```

Note that the default whitespace token is added to the start and end of the input tokens.

Tokens can be more than one character, and longer tokens are matched first:

```
yaml_ = """
  tokens:
    a: []      # "a" token with no classes
    aa: []     # "aa" token with no classes
    ' ': [wb]  # " " token and its class ("wb", for wordbreak)
  rules:
    aa: <DOUBLE_A>  # transliterate "aa" to "<DOUBLE_A>"
    a: <SINGLE_A>   # transliterate "a" to "<SINGLE_A>"
  whitespace:
    default: " "        # default whitespace token
    consolidate: false  # whitespace should not be consolidated
    token_class: wb     # whitespace token class
"""
gt_two = GraphTransliterator.from_yaml(yaml_)
gt_two.transliterate('a')
```

```
'<SINGLE_A>'
```

```
37 gt_two.transliterate('aa')
```

```
'<DOUBLE_A>'
```

```
38 gt_two.transliterate('aaa')
```

```
'<DOUBLE_A><SINGLE_A>'
```

Here the input "aaa" is transliterated as "<DOUBLE_A><SINGLE_A>", as the longer token "aa" is matched before "a".

Tokens can be assigned zero or more classes. Each class is a string of your choice. These classes are used in transliteration rules. In YAML they are defined as a dictionary, but internally the rules are stored as a dictionary of token strings keyed to a set of token classes. They can be accessed using `GraphTransliterator.tokens`:

```
39 gt_two.tokens
```

```
{'a': set(), 'aa': set(), ' ': {'wb'}}
```

Transliteration Rules

Graph Transliterator can handle a variety of transliteration tasks. To do so, it uses transliteration rules that contain **match settings** for particular tokens in specific contexts and also a resulting **production**, or string to be appended to the output string.

Match Settings

Transliteration rules contain the following parameters (ordered by where they would appear in a list of tokens):

- **previous token classes** : a list of token classes (optional)
- **previous tokens** : a list of tokens (optional)
- **tokens** : a list of tokens
- **next tokens** : a list of tokens (optional)
- **next token classes** : a list of token classes (optional)

One or more (**tokens**) must be matched in a particular location. However, specific tokens can be required before (**previous tokens**) or behind (**next tokens**) those tokens. Additionally, particular token classes can be required before (**previous token classes**) and behind (**next token classes**) all of the specific tokens required (previous tokens, tokens, next tokens).

Depending on their complexity, these match conditions can be entered using the "easy reading" format in the following ways.

If there are no required lookahead or lookbehind tokens, the rule can be as follows:

```
rules:
   a a: aa  # two tokens (a,a), with production "production_aa"
```

If, in an addition to tokens, there are specific previous or following tokens that must be matched, the rule can be entered as:

```
tokens:
  a: []
  b: []
  c: []
  d: []
rules:
   a (b): a_before_b  # matches  token 'a' with the next token 'b'
   (c) a: a_after_c   # matches token 'a' when the previous token is 'c'
   a (b c): a_before_b_and_c # matches token 'a' when next tokens are 'b' then 'c'
   (d) a (b c): a_after_d_and_before_b,c  # matches the token 'a' after 'd' and␣
↪before 'b' and 'c'
```

Token class names are indicated between angular brackets ("<classname>"). If preceding and following tokens are not required but classes are, these can be entered as follows:

```
tokens:
  a: []
  b: [class_b]
  c: []
  ' ': [wb]
rules:
  c <class_b>: c_after _class_b  # match token 'c' before a token of class 'class_b`
  <class_b> a: a_before_class_b  # match token 'a' after a token of class `class_b`
  <class_b> a <class_b>: a_between_class_b #  match token 'a' between tokens of class
↪'class_b'
```

If token classes must precede or follow specific tokens, these can be entered as:

```
tokens:
  a: []
  b: []
  c: [class_c]
  d: [class_d]
  ' ': [wb]
rules:
  d (b <class_c>): a_before_b_and_class_c # match token 'd' before 'b' and a token of␣
↪class 'class_c'
  (<class_c> b) a: a_after_b_and_class_c  # match token 'a' after 'b' and a token of␣
↪class 'class_c'
  (<class_c> d) a (b <class_c> <class_d>): x # match 'a' after token of 'class_c' and
↪'d' and before a token of 'class_c' and of 'class_d'
whitespace:
  default: ' '
  token_class: wb
  consolidate: false
```

Automatic Ordering of Transliteration Rules

Graph Transliterator automatically orders the transliteration rules based on the number of tokens required by the rule. It *picks the rule requiring the longest match in a given context*. It does so by assigning a cost to each transliteration rule that decreases depending on the number of tokens required by the rule. More tokens decreases the cost of a rule causing it to be matched first:

```
yaml_ = """
  tokens:
```

(continues on next page)

(continued from previous page)

```
    a: []
    b: []
    c: [class_of_c]
    ' ': [wb]
  rules:
    a: <<A>>
    a b: <<AB>>
    b: <<B>>
    c: <<C>>
    ' ': _
    <class_of_c> a b: <<AB_after_C>>
  whitespace:
    default: " "
    consolidate: false
    token_class: wb
"""
gt_three = GraphTransliterator.from_yaml(yaml_)
gt_three.transliterate("ab")  # should match rule "a b"
```

```
'<<AB>>'
```

```
gt_three.transliterate("cab") # should match rules: "c", and "<class_of_c> a b"
```

```
'<<C>><<AB_after_C>>'
```

Internally, Graph Transliterator uses a special `TransliterationRule` class. These can be accessed using `GraphTransliterator.rules`. Rules are sorted by cost, lowest to highest:

```
gt_three.rules
```

```
[TransliterationRule(production='<<AB_after_C>>', prev_classes=['class_of_c'], prev_
↪tokens=None, tokens=['a', 'b'], next_tokens=None, next_classes=None, cost=0.
↪32192809488736235),
 TransliterationRule(production='<<AB>>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a', 'b'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='<<A>>', prev_classes=None, prev_tokens=None, tokens=[
↪'a'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='<<B>>', prev_classes=None, prev_tokens=None, tokens=[
↪'b'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='<<C>>', prev_classes=None, prev_tokens=None, tokens=[
↪'c'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='_', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

Whitespace Settings

Whitespace is often very important in transliteration tasks, as the form of many letters may change at the start or end of words, as in the right-to-left Perso-Arabic and left-to-right Indic scripts. Therefore, Graph Transliterator requires the following **whitespace settings**:

- the **default** whitespace token
- the whitespace **token class**
- whether or not to **consolidate** whitespace

A whitespace token and token class must be defined for any Graph Transliterator. A whitespace character is added temporarily to the start and end of the input tokens during the transliteration process.

The `consolidate` option may be useful in particular transliteration tasks. It replaces any sequential whitespace tokens in the input string with the default whitespace character. At the start and end of input, it removes any whitespace:

```
yaml_ = """
  tokens:
    a: []
    ' ': [wb]
  rules:
    <wb> a: _A
    a <wb>: A_
    <wb> a <wb>: _A_
    a: a
    ' ': ' '
  whitespace:
    default: " "        # default whitespace token
    consolidate: true   # whitespace should be consolidated
    token_class: wb     # whitespace token class
"""
gt = GraphTransliterator.from_yaml(yaml_)
gt.transliterate('a')   # whitespace present at start of string
```

```
'_A_'
```

```
gt.transliterate('aa')  # whitespace present at start and end of string
```

```
'_AA_'
```

```
gt.transliterate(' a')  # consolidate removes whitespace at start of string
```

```
'_A_'
```

```
gt.transliterate('a ')  # consolidate removes whitespace at end of string
```

```
'_A_'
```

Whitespace settings are stored internally as `WhitespaceRules` and can be accessed using `GraphTransliterator.whitespace`:

```
gt.whitespace
```

```
WhitespaceRules(default=' ', token_class='wb', consolidate=True)
```

On Match Rules

Graph Transliterator allows strings to be inserted right before the productions of transliteration rules. These take as parameters:

- a list of **previous token classes**, preceding the location of the transliteration rule match
- a list of **next token classes**, from the index of the transliteration rule match
- a **production** string to insert

In the easy reading YAML format, the `onmatch_rules` are a list of dictionaries. The key consists of the token class names in angular brackets ("<classname>"), and the previous classes to match are separated from the following classes by a "+". The production is the value of the dictionary:

```
yaml_ = """
  tokens:
    a: [vowel]
    ' ': [wb]
  rules:
    a: A
    ' ': ' '
  whitespace:
    default: " "
    consolidate: false
    token_class: wb
  onmatch_rules:
    - <vowel> + <vowel>: ',' # add a comma between vowels
 """
gt = GraphTransliterator.from_yaml(yaml_)
gt.transliterate('aa')
```

```
'A,A'
```

On Match rules are stored internally as a `OnMatchRule` and can be accessed using `GraphTransliterator.onmatch_rules`:

```
gt.onmatch_rules
```

```
[OnMatchRule(prev_classes=['vowel'], next_classes=['vowel'], production=',')]
```

Metadata

Graph Transliterator allows for the storage of metadata as another input parameter, `metadata`. It is a dictionary, and fields can be added to it:

```
yaml_ = """
  tokens:
    a: []
    ' ': [wb]
  rules:
    a: A
```

(continues on next page)

(continued from previous page)

```
    ' ': ' '
  whitespace:
    default: " "
    consolidate: false
    token_class: wb
  metadata:
    author: Author McAuthorson
    version: 0.1.1
    description: A sample Graph Transliterator
  """
gt = GraphTransliterator.from_yaml(yaml_)
gt.metadata
```

```
{'author': 'Author McAuthorson',
 'version': '0.1.1',
 'description': 'A sample Graph Transliterator'}
```

Unicode Support

Graph Transliterator allows Unicode characters to be specified by name, including in YAML files, using the format "\N{UNICODE CHARACTER NAME}" or "\u{####}" (where #### is the hexadecimal character code):

```
yaml_ = """
  tokens:
    b: []
    c: []
    ' ': [wb]
  rules:
    b: \N{LATIN CAPITAL LETTER B}
    c: \u0043    # hexadecimal Unicode character code for 'C'
    ' ': ' '
  whitespace:
    default: " "
    consolidate: false
    token_class: wb
  """
gt = GraphTransliterator.from_yaml(yaml_)
gt.transliterate('b')
```

```
'B'
```

```
gt.transliterate('c')
```

```
'C'
```

Configuring Directly

In addition to using `GraphTansliterator.from_yaml()` and `GraphTransliterator.from_yaml_file()`, Graph Transliterator can also be configured and initialized directly using basic Python types passed as dictionary to `GraphTransliterator.from_dict()`

```
settings = {
  'tokens': {'a': ['vowel'],
             ' ': ['wb']},
  'rules': [
      {'production': 'A', 'tokens': ['a']},
      {'production': ' ', 'tokens': [' ']}],
  'onmatch_rules': [
      {'prev_classes': ['vowel'],
       'next_classes': ['vowel'],
       'production': ','}],
  'whitespace': {
      'default': ' ',
      'consolidate': False,
      'token_class': 'wb'},
  'metadata': {
      'author': 'Author McAuthorson'}
}
gt = GraphTransliterator.from_dict(settings)
gt.transliterate('a')
```

```
'A'
```

This feature can be useful if generating a Graph Transliterator using code as opposed to a configuration file.

Ambiguity Checking

Graph Transliterator, by default, will check for ambiguity in its transliteration rules. If two rules of the same cost would match the same string(s) and those strings would not be matched by a less costly rule, an `AmbiguousTransliterationRulesException` occurs. Details of all exceptions will be reported as a `logging.warning()`:

```
yaml_ = """
tokens:
  a: [class1, class2]
  b: []
  ' ': [wb]
rules:
  <class1> a: A
  <class2> a: AA # ambiguous rule
  <class1> b: BB
  b <class2>: BB # also ambiguous
whitespace:
  default: ' '
  consolidate: True
  token_class: wb
"""
gt = GraphTransliterator.from_yaml(yaml_)
```

```
WARNING:root:The pattern [{'a'}, {'a'}, {' ', 'b', 'a'}] can be matched by␣
↪both:
  <class1> a
```

```
  <class2> a


WARNING:root:The pattern [{'a'}, {'b'}, {'a'}] can be matched by both:
  <class1> b
  b <class2>
```

```
AmbiguousTransliterationRulesException
```

The warning shows the set of possible previous tokens, matched tokens, and next tokens as three sets.

Ambiguity checking is only necessary when using an untested Graph Transliterator. It can be turned off during initialization. To do so, set the initialization parameter `check_ambiguity` to *False*.

Ambiguity checking can also be done on demand using `check_for_ambiguity()`.

Ambiguity checking is not performed if loading from a serialized GraphTransliterator using `GraphTransliterator.load()` or `GraphTransliterator.loads()`.

Setup Validation

Graph Transliterator validates both the "easy reading" configuration and the direct configuration using the `marshmallow` library.

Transliteration and Its Exceptions

The main method of Graph Transliterator is `GraphTransliterator.transliterate()`. It will return a string:

```
GraphTransliterator.from_yaml(
'''
tokens:
  a: []
  ' ': [wb]
rules:
  a: A
  ' ': '_'
whitespace:
  default: ' '
  consolidate: True
  token_class: wb
''').transliterate("a a")
```

```
'A_A'
```

Details of transliteration error exceptions will be logged using `logging.warning()`.

Unrecognizable Input Token

Unless the `GraphTransliterator` is initialized with or has the property `ignore_errors` set as `True`, `GraphTransliterator.transliterate()` will raise `UnrecognizableInputTokenException` when character(s) in the input string do not correspond to any defined types of input tokens. In both cases, there will be a `logging.warning()`:

```
from graphtransliterator import GraphTransliterator
yaml_ = """
  tokens:
   a: []
   ' ': [wb]
  rules:
    a: A
    ' ': ' '
  whitespace:
    default: " "
    consolidate: true
    token_class: wb
"""
GraphTransliterator.from_yaml(yaml_).transliterate("a!a") # ignore_errors=False
```

```
WARNING:graphtransliterator:Unrecognizable token ! at pos 1 of a!a
```

```
UnrecognizableInputTokenException
```

```
GraphTransliterator.from_yaml(yaml_, ignore_errors=True).transliterate("a!a") #␣
↪ignore_errors=True
```

```
WARNING:graphtransliterator:Unrecognizable token ! at pos 1 of a!a
```

```
'AA'
```

No Matching Transliteration Rule

Another possible error occurs when no transliteration rule can be identified at a particular index in the index string. In that case, there will be a `logging.warning()`. If the parameter `ignore_errors` is set to `True`, the token index will be advanced. Otherwise, there will be a `NoMatchingTransliterationRuleException`:

```
yaml_='''
  tokens:
    a: []
    b: []
    ' ': [wb]
  rules:
    a: A
    b (a): B
  whitespace:
    default: ' '
    token_class: wb
    consolidate: False
'''
```

(continues on next page)

(continued from previous page)

```
gt = GraphTransliterator.from_yaml(yaml_)
gt.transliterate("ab")
```

```
WARNING:graphtransliterator:No matching transliteration rule at token pos 2␣
↪of [' ', 'a', 'b', ' ']
```

```
NoMatchingTransliterationRuleException
```

```
gt.ignore_errors = True
gt.transliterate("ab")
```

```
WARNING:graphtransliterator:No matching transliteration rule at token pos 2␣
↪of [' ', 'a', 'b', ' ']
```

```
'A'
```

Additional Methods

Graph Transliterator also offers a few additional methods that may be useful for particular tasks.

Serialization and Deserialization

The settings of a Graph Transliterator can be serialized using `GraphTransliterator.dump()`, which returns a dictionary of native Python data types. A JSON string of the same can be accessed using `GraphTransliterator.dumps()`. Validation is not performed during a dump.

By default, `GraphTransliterator.dumps()` will use compression level 2, which removes the internal graph and indexes tokens and graph node labels. Compression level 1 also indexes tokens and graph node labels and contains the graph. Compression level 0 is human readable and includes the graph. No information is lost during compression. Level 2, the default, loads the fastest and also has the smallest file size.

A GraphTransliterator can be loaded from serialized settings, e.g. in an API context, using `GraphTransliterator.load()` and from JSON data as `GraphTransliterator.loads()`. Because they are intended to be quick, neither method performs ambiguity checks or strict validation checking by default.

Serialization can be useful if providing an API or making the configured Graph Transliterator available in other programming languages, e.g. Javascript.

Matching at an Index

The method `match_at()` is also public. It matches the best transliteration rule at a particular index, which is the rule that contains the largest number of required tokens. The method also has the option `match_all` which, if set, returns all possible transliteration matches at a particular location:

```
gt = GraphTransliterator.from_yaml('''
        tokens:
            a: []
            a a: []
            ' ': [wb]
```

(continues on next page)

(continued from previous page)

```
        rules:
            a: <A>
            a a: <AA>
        whitespace:
            default: ' '
            consolidate: True
            token_class: wb
''')
tokens = gt.tokenize("aa")
tokens # whitespace added to ends
```

```
[' ', 'a', 'a', ' ']
```

```
gt.match_at(1, tokens) # returns index to rule
```

```
0
```

```
gt.rules[gt.match_at(1, tokens)] # actual rule
```

```
TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None, tokens=['a
↪', 'a'], next_tokens=None, next_classes=None, cost=0.41503749927884376)
```

```
gt.match_at(1, tokens, match_all=True) # index to rules, with match_all
```

```
[0, 1]
```

```
[gt.rules[_] for _ in gt.match_at(1, tokens, match_all=True)] # actual rules, with␣
↪match_all
```

```
[TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None, tokens=[
↪'a', 'a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None, tokens=['a
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

Details of Matches

Each Graph Transliterator has a property `last_matched_rules` which returns a list of `TransliterationRule` of the previously matched transliteration rules:

```
gt.transliterate("aaa")
```

```
'<AA><A>'
```

```
gt.last_matched_rules
```

```
[TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None, tokens=[
↪'a', 'a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None, tokens=['a
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

The particular tokens matched by those rules can be accessed using `last_matched_rule_tokens`:

```
gt.last_matched_rule_tokens
```

```
[['a', 'a'], ['a']]
```

Pruning of Rules

In particular cases, it may be useful to remove certain transliteration rules from a more robustly defined Graph Transliterator based on the string output produced by the rules. That can be done using `pruned_of()`:

```
gt.rules
```

```
[TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None, tokens=[
↪'a', 'a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None, tokens=['a
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

```
gt.pruned_of('<AA>').rules
```

```
[TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None, tokens=['a
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

```
gt.pruned_of(['<A>', '<AA>']).rules
```

```
[]
```

Internal Graph

Graph Transliterator creates a directed tree during its initialization. During calls to `transliterate()`, it searches that graph to find the best transliteration match at a particular index in the tokens of the input string.

DirectedGraph

The tree is an instance of `DirectedGraph` that can be accessed using `GraphTransliterator.graph`. It contains: a list of nodes, each consisting of a dictionary of attributes; a dictionary of edges keyed between the head and tail of an edge that contains a dictionary of edge attributes; and finally an edge list.

```
gt = GraphTransliterator.from_yaml(
    """
    tokens:
      a: []
      ' ': [wb]
    rules:
      a: b
      <wb> a: B
      ' ': ' '
    whitespace:
      token_class: wb
      default: ' '
      consolidate: false
```

(continues on next page)

(continued from previous page)

```
    """)
gt.graph
```

```
<graphtransliterator.graphs.DirectedGraph at 0x7fd57c312b00>
```

Nodes

The tree has nodes of three types: *Start*, *token*, and *rule*. A single *Start* node, the root, is connected to all other nodes. A *token* node corresponds to a token having been matched. Finally, *rule* nodes are leaf nodes (with no outgoing edges) that correspond to matched transliteration rules:

```
gt.graph.node
```

```
[{'type': 'Start', 'ordered_children': {'a': [1], ' ': [4]}},
 {'type': 'token', 'token': 'a', 'ordered_children': {'__rules__': [2, 3]}},
 {'type': 'rule', 'rule_key': 0, 'accepting': True, 'ordered_children': {}},
 {'type': 'rule', 'rule_key': 1, 'accepting': True, 'ordered_children': {}},
 {'type': 'token', 'token': ' ', 'ordered_children': {'__rules__': [5]}},
 {'type': 'rule', 'rule_key': 2, 'accepting': True, 'ordered_children': {}}]
```

Edges

Edges between these nodes may have different constraints in their attributes:

```
gt.graph.edge
```

```
{0: {1: {'token': 'a', 'cost': 0.41503749927884376},
  4: {'token': ' ', 'cost': 0.5849625007211562}},
 1: {2: {'cost': 0.41503749927884376, 'constraints': {'prev_classes': ['wb']}},
  3: {'cost': 0.5849625007211562}},
 4: {5: {'cost': 0.5849625007211562}}}
```

Before the *token* nodes, there is a *token* constraint on the edge that must be matched before the transliterator can visit the token node:

```
gt.graph.edge[0][1]
```

```
{'token': 'a', 'cost': 0.41503749927884376}
```

On the edges before rules there may be other *constraints*, such as certain tokens preceding or following tokens of the corresponding transliteration rule:

```
gt.graph.edge[1][2]
```

```
{'cost': 0.41503749927884376, 'constraints': {'prev_classes': ['wb']}}
```

An edge list is also maintained that consists of a tuple of (head, tail):

```
gt.graph.edge_list
```

```
[(0, 1), (1, 2), (1, 3), (0, 4), (4, 5)]
```

Search and Preprocessing

Graph Transliterator uses a best-first search, implemented using a stack, that finds the transliteration with the the lowest cost. The cost function is:

$$\text{cost}(rule) = \log_2 \left(1 + \frac{1}{1 + \text{count\_of\_tokens\_in}(rule)}\right)$$

It results in a number between 1 and 0 that lessens as more tokens must be matched. Each edge on the graph has a cost attribute that is set to the lowest cost transliteration rule following it. When transliterating, Graph Transliterator will try lower cost edges first and will backtrack if the constraint conditions are not met.

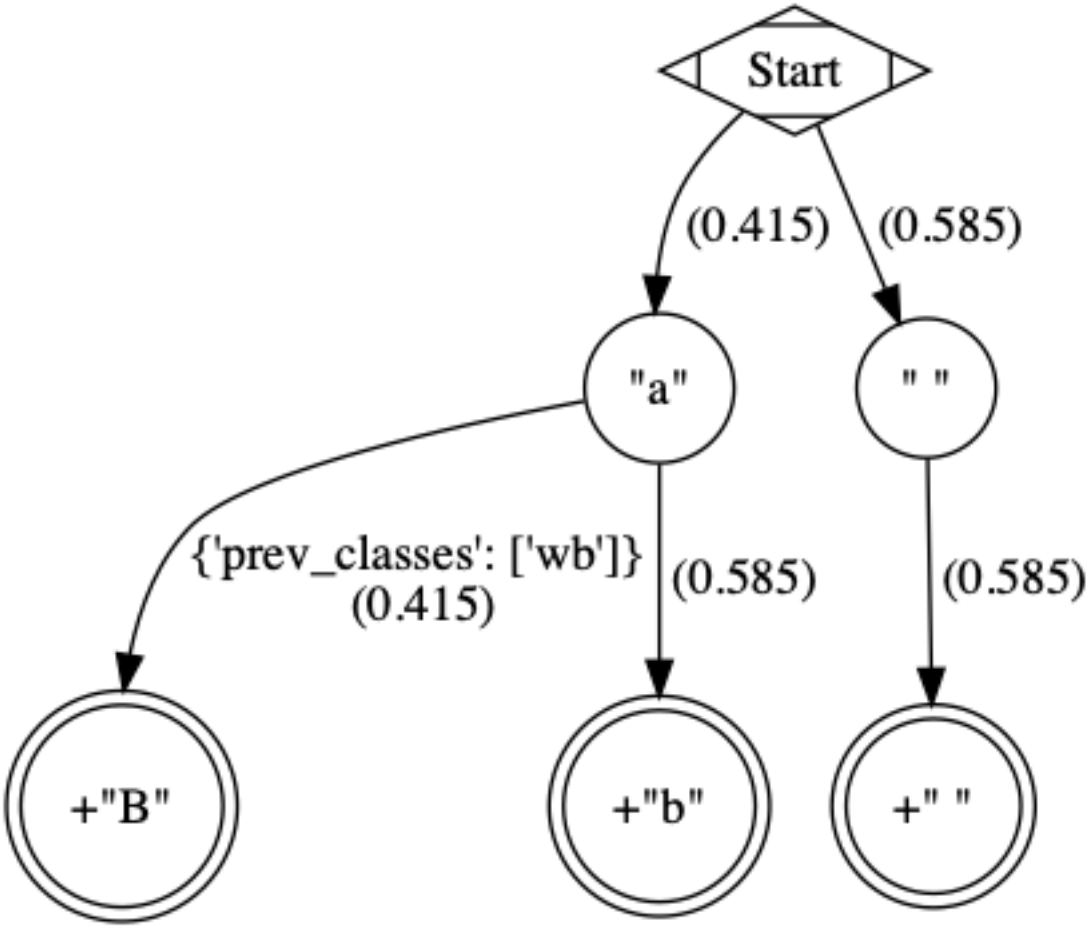


Fig. 2: An example graph created for the simple case of a Graph Transliterator that takes as input two token types, `a` and `" "` (space), and renders `" "` as `" "`, and `a` as `b` unless it follows a token of class `wb` (for wordbreak), in which case it renders `a` as `B`. The *rule* nodes are in double circles, and *token* nodes are single circles. The numbers are the cost of the particular edge, and less costly edges are searched first. Previous token class (`prev_classes`) constraints are found on the edge before the leftmost leaf rule node.

To optimize the search, during initialization an `ordered_children` dictionary is added to each non-leaf node. Its values are a list of node indexes sorted by cost and keyed by the following *token*:

```
gt.graph.node[0]
```

```
{'type': 'Start', 'ordered_children': {'a': [1], ' ': [4]}}
```

Any *rule* connected to a node is added to each *ordered_children*. Any rule nodes immediately following the current node are keyed to `__rules__`:

```
gt.graph.node[1]
```

```
{'type': 'token', 'token': 'a', 'ordered_children': {'__rules__': [2, 3]}}
```

Because of this preprocessing, Graph Transliterator does not need to iterate through all of the outgoing edges of a node to find the next node to search.

3.2.3 Bundled Transliterators

Note: Python code on this page: `bundled.py` Jupyter Notebook: `bundled.ipynb`

Graph Transliterator includes bundled transliterators in a `Bundled` subclass of `GraphTransliterator` that can be used as follows:

```
import graphtransliterator.transliterators as transliterators
example_transliterator = transliterators.Example()
example_transliterator.transliterate('a')
```

```
'A'
```

To access transliterator classes, use the iterator `transliterators.iter_transliterators()`:

```
bundled_iterator = transliterators.iter_transliterators()
next(bundled_iterator)
```

```
<example.Example at 0x7f34c0678210>
```

To access the names of transliterator classes, use the iterator `transliterators.iter_names()`:

```
bundled_names_iterator = transliterators.iter_names()
next(bundled_names_iterator)
```

```
'Example'
```

The actual bundled transliterators are submodules of *graphtransliterator.transliterators*, but they are loaded into the namespace of `transliterators`:

```
from graphtransliterator.transliterators import Example
```

Each instance of `Bundled` contains a `directory` attribute:

```
transliterator = Example()
transliterator.directory
```

```
'/home/docs/checkouts/readthedocs.org/user_builds/graphtransliterator/checkouts/
↪latest/graphtransliterator/transliterators/example'
```

Each will contain an easy-reading YAML file that you can view:

```
tokens:
  a: [vowel]
  ' ': [whitespace]
  b: [consonant]
rules:
  a: A
  b: B
  ' ': ' '
  (<consonant> a) b (a <consonant>):  "!B!"
onmatch_rules:
  - <vowel> + <vowel>: ","
whitespace:
  consolidate: False
  default: " "
  token_class: whitespace
metadata:
  name: example
  version: 1.0.0
  description: "An Example Bundled Transliterator"
  url: https://github.com/seanpue/graphtransliterator/tree/master/transliterator/
↪sample
  author: Author McAuthorson
  author_email: author_mcauthorson@msu.edu
  license: MIT License
  keywords:
    - example
  project_urls:
    Documentation: https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example
    Source: https://github.com/seanpue/graphtransliterator/tree/graphtransliterator/
↪transliterators/example
    Tracker: https://github.com/seanpue/graphtransliterator/issues
```

There is also a JSON dump of the transliterator for quick loading:

```
{"graphtransliterator_version":"1.2.0","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],","]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},null]}
```

Test Coverage of Bundled Transliterators

Each bundled transliterators requires rigorous testing: every node and edge, as well as any onmatch rules, if applicable, must be visited. A separate subclass `CoverageTransliterator` of `GraphTransliterator` is used during testing.

It logs visits to nodes, edges, and onmatch rules. The tests are found in a subdirectory of the transliterator named "tests". They are in a YAML file consisting of a dictionary keyed from transliteration input to correct output, e.g.:

```
# YAML declaration of tests for bundled Graph Transliterator
# These are in the form of a dictionary.
# The key is the source text, and the value is the correct transliteration.
' ': ' '
a: A
aa: A,A
babab: BA!B!AB
b: B
```

Once the tests are completed, Graph Transliterator checks that all components of the graph and all of the onmatch rules have been visited.

Class Structure and Naming Conventions

Each transliterator must include a class definition in a submodule of `transliterators`.

The class name of each transliterator must be unique and follow camel-case conventions, e.g. *SourceToTarget*. File and directory names should, if applicable, be lowercased as *source_to_target*.

The bundled files should follow this directory structure, where {{source_to_target}} is the name of the transliterator:

```
transliterators
├── {{source_to_target}}
|   ├── __init__.py
|   ├── {{source_to_target}}.json
|   ├── {{source_to_target}}.yaml
└── tests
    ├── test_{{source_to_target}}.py
    └── {{source_to_target}}_tests.yaml
```

The bundled transliterator will:

- include both an easy-reading YAML file `{{source_to_target}}.yaml` and a JSON file `{{source_to_target}}.json`.
- have tests in a YAML format consisting of a dictionary keyed from transliteration to correct output in `{{source_to_target}}_tests.yaml`. It must include complete test coverage of its graph. Every node and edge of the graph must be visited during the course of the tests, as well as every on-match rule. Each on-match rule must be utilized during the course of the tests.
- include metadata about the transliterator in its easy-reading YAML file.
- have an optional custom test file `test_{{source_to_target.py}}`. This is useful during development.

Metadata Requirements

Each `Bundled` transliterator can include the following metadata fields. These fields are a subset of the metadata of `setuptools`.

name (*str*)
Name of the transliterator, e.g. "source_to_target".

version (*str*, optional)
Version of the transliterator. Semantic versioning (https://semver.org) is recommended.

url (*str*, optional)
URL for the transliterator, e.g. github repository.

author (*str*, optional)
Author of the transliterator

author_email (*str*, optional)
E-mail address of the author.

maintainer (*str*, optional)
Name of the maintainer.

maintainer_email (*str*, optional)
E-mail address of the maintainer.

license (*str*, optional)
License of the transliterator. An open-source license is required for inclusion in this project.

keywords (*list* of *str*, optional)
List of keywords.

project_urls (*dict* of {*str*: *str*}, optional)
Dictionary of project URLS, e.g. *Documentation*, *Source*, etc.

Metadata is validated using a `BundledMetadataSchema` found in `transliterators.schemas`.

To browse metadata, you can use `iter_transliterators()`:

```
import pprint
transliterator = next(transliterators.iter_transliterators())
pprint.pprint(transliterator.metadata)
```

```
{'author': 'Author McAuthorson',
 'author_email': 'author_mcauthorson@msu.edu',
 'description': 'An Example Bundled Transliterator',
 'keywords': ['example'],
 'license': 'MIT License',
 'name': 'example',
 'project_urls': {'Documentation': 'https://github.com/seanpue/graphtransliterator/
↪tree/master/graphtransliterator/transliterators/example',
                  'Source': 'https://github.com/seanpue/graphtransliterator/tree/
↪graphtransliterator/transliterators/example',
                  'Tracker': 'https://github.com/seanpue/graphtransliterator/issues'},
 'url': 'https://github.com/seanpue/graphtransliterator/tree/master/transliterator/
↪sample',
 'version': '1.0.0'}
```

3.2.4 Command Line Interface

Graph Transliterator has a simple command line interface with six commands: `dump`, `dump-tests`, `generate-tests`, `list-bundled`, `make-json`, `test`, and `transliterate`.

```
$ graphtransliterator --help
```

```
Usage: main [OPTIONS] COMMAND [ARGS]...

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.

Commands:
  dump            Dump transliterator as JSON.
  dump-tests      Dump BUNDLED tests.
  generate-tests  Generate tests as YAML.
  list-bundled    List BUNDLED transliterators.
  make-json       Make JSON rules of BUNDLED transliterator(s).
  test            Test BUNDLED transliterator.
  transliterate   Transliterate INPUT.
```

Dump

The `dump` command will output the specified transliterator as JSON:

```
$ graphtransliterator dump --help
```

```
Usage: dump [OPTIONS]

  Dump transliterator as JSON.

Options:
  -f, --from <CHOICE TEXT>...     Format (bundled/yaml_file) and source (name or
                                  filename) of transliterator  [required]
  -ca, --check-ambiguity / -nca, --no-check-ambiguity
                                  Check for ambiguity.  [default: no-check-
                                  ambiguity]
  -cl, --compression-level INTEGER
                                  Compression level, from 0 to 2  [default: 2]
  --help                          Show this message and exit.
```

It require a `--from` or `-f` option with two arguments. The first argument specifies the format of the transliterator (*bundled* or *yaml_file*) and the second a parameter for that format (the name of the bundled transliterator or the name of a YAML file).

To load a bundled transliterator, used *bundled* as the first parameter and give its (class) name, which will be in CamelCase, as the second:

```
$ graphtransliterator dump --from bundled Example
```

```
{"graphtransliterator_version":"1.2.4","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
```

(continues on next page)

(continued from previous page)

```
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],",",]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},null]}
```

To load from a YAML file, give *yaml_file* as the first and the the name of the file as the second parameter:

```
$ graphtransliterator dump --from yaml_file ../graphtransliterator/transliterators/
↪example/example.yaml
```

```
{"graphtransliterator_version":"1.2.4","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],",",]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},null]}
```

If you want to check for ambiguity in the transliterator before the dump, use the `--check-ambiguity` or `-ca` option:

```
$ graphtransliterator dump --from bundled Example --check-ambiguity # human readable
```

```
{"graphtransliterator_version":"1.2.4","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],",",]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},null]}
```

The compression level can of the JSON be specified using the `--compression-level` or `-cl` command. Compression level 0 is human readable; compression level 1 is not human readable and includes the generated graph; compression level 2 is not human readable and does not include the graph. Compression level 2, which is the fastest, is the default. There is no information lost during these compressions:

```
$ graphtransliterator dump --from bundled Example --compression-level 0 # human␣
↪readable, with graph
```

```
{"tokens": {"a": ["vowel"], " ": ["whitespace"], "b": ["consonant"]}, "rules": [{
↪"production": "!B!", "prev_classes": ["consonant"], "prev_tokens": ["a"], "tokens":␣
↪["b"], "next_classes": ["consonant"], "next_tokens": ["a"], "cost": 0.
↪22239242133644802}, {"production": "A", "tokens": ["a"], "cost": 0.5849625007211562}
↪, {"production": "B", "tokens": ["b"], "cost": 0.5849625007211562}, {"production":
↪" ", "tokens": [" "], "cost": 0.5849625007211562}], "whitespace": {"default": " ",
↪"token_class": "whitespace", "consolidate": false}, "onmatch_rules": [{"prev_classes
↪": ["vowel"], "next_classes": ["vowel"], "production": ","}], "metadata": {"name":
↪"example", "version": "1.0.0", "description": "An Example Bundled Transliterator",
↪"url": "https://github.com/seanpue/graphtransliterator/tree/master/transliterator/
↪sample", "author": "Author McAuthorson", "author_email": "author_mcauthorson@msu.edu
↪", "license": "MIT License", "keywords": ["example"], "project_urls": {
↪"Documentation": "https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example", "Source": "https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example", "Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}}, "ignore_errors": false,
↪"onmatch_rules_lookup": {"a": {"a": [0]}}, "tokens_by_class": {"vowel": ["a"],
↪"whitespace": [" "], "consonant": ["b"]}, "graph": {"node": [{"type": "Start",
↪"ordered_children": {"b": [1], "a": [3], " ": [6]}}, {"token": "b", "type": "token",
↪ "ordered_children": {"__rules__": [2, 5]}}, {"type": "rule", "accepting": true,
↪"rule_key": 0}, {"token": "a", "type": "token", "ordered_children": {"__rules__":␣
↪[4]}}, {"type": "rule", "accepting": true, "rule_key": 1}, {"type": "rule",
↪"accepting": true, "rule_key": 2}, {"token": " ", "type": "token", "ordered_children
↪": {"__rules__": [7]}}, {"type": "rule", "accepting": true, "rule_key": 3}], "edge
↪": {"0": {"1": {"token": "b", "cost": 0.22239242133644802}, "3": {"token": "a",
↪"cost": 0.5849625007211562}, "6": {"token": " ", "cost": 0.5849625007211562}}, "1":
↪{"2": {"cost": 0.22239242133644802, "constraints": {"prev_classes": ["consonant"],
↪"prev_tokens": ["a"], "next_tokens": ["a"], "next_classes": ["consonant"]}}, "5": {
↪"cost": 0.5849625007211562}}, "3": {"4": {"cost": 0.5849625007211562}}, "6": {"7": {
↪"cost": 0.5849625007211562}}}, "edge_list": [[0, 1], [0, 3], [0, 6], [1, 2], [1, 5],
↪ [3, 4], [6, 7]]}, "tokenizer_pattern": "(b|a|\\ )", "graphtransliterator_version":
↪"1.2.4"}
```


```
$ graphtransliterator dump --from bundled Example --compression-level 1 # not human␣
↪readable, with graph
```


```
{"graphtransliterator_version":"1.2.4","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],","]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},[["Start","rule","token"],
↪[[0,0,{"2":[1],"1":[3],"0":[6]}],[2,0,2,{"-1":[2,5]}],[1,1,0],[2,0,1,{"-1":[4]}],[1,
↪1,1],[1,1,2],[2,0,0,{"-1":[7]}],[1,1,3]],{"0":{"1":[0,-5,2],"3":[0,-1,1],"6":[0,-1,
↪0]},"1":{"2":[[[0],[1],[1],[0]],-5,-1],"5":[0,-1,-1]},"3":{"4":[0,-1,-1]},"6":{"7
↪":[0,-1,-1]}}]]}
```


```
$ graphtransliterator dump --from bundled Example --compression-level 2 # default;␣
↪not human readable, no graph
```

```
{"graphtransliterator_version":"1.2.4","compressed_settings":[["consonant","vowel",
↪"whitespace"],[" ","a","b"],[[2],[1],[0]],[["!B!",[0],[1],[2],[1],[0],-5],["A",0,0,
↪[1],0,0,-1],["B",0,0,[2],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","whitespace",0],[[[1],
↪[1],","]],{"name":"example","version":"1.0.0","description":"An Example Bundled␣
↪Transliterator","url":"https://github.com/seanpue/graphtransliterator/tree/master/
↪transliterator/sample","author":"Author McAuthorson","author_email":"author_
↪mcauthorson@msu.edu","license":"MIT License","keywords":["example"],"project_urls":{
↪"Documentation":"https://github.com/seanpue/graphtransliterator/tree/master/
↪graphtransliterator/transliterators/example","Source":"https://github.com/seanpue/
↪graphtransliterator/tree/graphtransliterator/transliterators/example","Tracker":
↪"https://github.com/seanpue/graphtransliterator/issues"}},null]}
```


Dump Tests

The `dump-tests` command dumps the tests of a bundled transliterator:

```
$ graphtransliterator dump-tests --help
```

```
Usage: dump-tests [OPTIONS] BUNDLED

  Dump BUNDLED tests.

Options:
  -t, --to [json|yaml]  Format (json/yaml) in which to dump  [default: yaml]
  --help                Show this message and exit.
```

By default, it outputs the original YAML tests file, preserving any comments:

```
$ graphtransliterator dump-tests Example
```

```
# YAML declaration of tests for bundled Graph Transliterator
# These are in the form of a dictionary.
# The key is the source text, and the value is the correct transliteration.
' ': ' '
a: A
aa: A,A
babab: BA!B!AB
b: B
```

To output as JSON, use the `--to` or `-t` flag:

```
$ graphtransliterator dump-tests --to json Example
```

```
{" ": " ", "a": "A", "aa": "A,A", "babab": "BA!B!AB", "b": "B"}
```

Generate Tests

The `generate-tests` command generates YAML tests keyed from input to desired output covering the entire internal graph. This command can be used to view the output of the transliterator in Unicode. It can also be used to generate starter tests for bundled transliterators:

```
$ graphtransliterator generate-tests --help
```

```
Usage: generate-tests [OPTIONS]

  Generate tests as YAML.

Options:
  -f, --from <CHOICE TEXT>...     Format (bundled/json/json_file/yaml_file) and
                                  source (name, JSON, or filename) of
                                  transliterator  [required]
  -ca, --check-ambiguity / -nca, --no-check-ambiguity
                                  Check for ambiguity.  [default: no-check-
                                  ambiguity]
  --help                          Show this message and exit.
```

It also require a `--from` or `-f` option with two arguments. The first argument specifies the format of the transliterator (*bundled*, *json*, *json_file*, *yaml_file*), and the second a parameter for that format (the name of the bundled transliterator, the actual JSON, or the name of a YAML file). Ambiguity checking can be turned on using `--check_ambiguity` or `-ca`:

```
$ graphtransliterator generate-tests --from bundled Example
```

```
' ': ' '
a: A
aa: A,A
b: B
babab: BA!B!AB
```

List Bundled Transliterators

The `list-bundled` command provides a list of bundled transliterators:

```
$ graphtransliterator test --help
```

Make JSON of Bundled Transliterator(s)

The `make-json` command makes new JSON files of bundled transliterators:

```
$ graphtransliterator make-json --help
```

It also allows regular-expression matching using the `--reg-ex` or `-re` flag. Matching starts at the start of the string. This command is for people creating new bundled transliterators.

Test

The `test` command tests a bundled transliterator:

```
$ graphtransliterator test --help
```

```
Usage: test [OPTIONS] BUNDLED

  Test BUNDLED transliterator.

Options:
  -ca, --check-ambiguity / -nca, --no-check-ambiguity
                                  Check for ambiguity.  [default: no-check-
                                  ambiguity]
  --help                          Show this message and exit.
```

It can only be used with bundled transliterators, so it only needs the name of the transliterator as its argument. This feature is useful when developing a transliterator. You can write the tests first and then begin developing the transliterator:

```
$ graphtransliterator test Example
```

```
True
```

Transliterate

The `transliterate` command will transliterate any following arguments:

```
$ graphtransliterator transliterate --help
```

```
Usage: transliterate [OPTIONS] [INPUT]...

  Transliterate INPUT.

Options:
  -f, --from <CHOICE TEXT>...     Format (bundled/json/json_file/yaml_file) and
                                  source (name, JSON, or filename) of
                                  transliterator  [required]
  -t, --to [json|python]          Format in which to output  [default: python]
  -ca, --check-ambiguity / -nca, --no-check-ambiguity
                                  Check for ambiguity.  [default: no-check-
                                  ambiguity]
  -ie, -nie, --ignore-errors / --no-ignore-errors
                                  Ignore errors.  [default: no-ignore-errors]
  --help                          Show this message and exit.
```

It also requires a `--from` or `-f` option with two arguments. The first argument specifies the format of the transliterator (*bundled*, *json*, *json_file*, *yaml_file*), and the second a parameter for that format (the name of the bundled transliterator, the actual JSON, or the name of a YAML file).

The *transliterate* command will transliterate every argument that follows. If there is only one input string, it will return a string:

```
$ graphtransliterator transliterate --from bundled Example a
```

```
A
```

```
$ graphtransliterator transliterate -f json_file ../graphtransliterator/
↪transliterators/example/example.json a
```

```
A
```

```
$ graphtransliterator transliterate -f yaml_file ../graphtransliterator/
↪transliterators/example/example.yaml a
```

```
A
```

Otherwise, it will return a list:

```
$ graphtransliterator transliterate -f bundled Example a a
```

```
['A', 'A']
```

The *transliterate* command also an optional `--to` or `-t` command that specifies the output format, a `` `python `` string (default) or a `json` string:

```
$ graphtransliterator transliterate --from bundled Example a
```

```
A
```

```
$ graphtransliterator transliterate --from bundled Example --to json a
```

```
"A"
```

```

```

```
$ graphtransliterator transliterate --from bundled Example --to python a a
```

```
['A', 'A']
```

```
$ graphtransliterator transliterate --from bundled Example --to json a a
```

```
["A", "A"]
```

3.2.5 Tutorial: Using GraphTransliterator

Note: Python code on this page: `tutorial.py` Jupyter Notebook: `tutorial.ipynb`

Graph Transliterator is designed to allow you to quickly develop rules for transliterating between languages and scripts. In this tutorial you will use a portion of Graph Transliterators features, including its token matching, class-based matching, and on match rules, using the `GraphTransliterator` class.

Tutorial Overview

The task for this tutorial will be to design a transliterator between the ITRANS (Indian languages TRANSliteration) encoding for Devanagari (Hindi) and standard Unicode. ITRANS developed as a means to transliterate Indic-language using the latin alphabet and punctuation marks before there were Unicode fonts.

The Devanagari alphabet is an abugida (alphasyllabary), where each "syllable" is a separate symbol. Vowels, except for the default अ ("a") have a unique symbol that connects to a consonant. At the start of the words, they have a unique shape. Consonants in sequence, without intermediary vowels, change their shape and are joined together. In Unicode, that is accomplished by using the Virama character.

Graph Transliterator works by first converting the input text into a series of tokens. In this tutorial you will define the tokens of ITRANS and necessary token classes that will allow us to generate rules for conversion.

Graph Transliterator allows rule matching by preceding tokens, tokens, and following tokens. It allows token classes to precede or follow any specific tokens. For this task, you will use a preceding token class to identify when to write vowel signs as opposed to full vowel characters.

Graph Transliterator also allows the insertion of strings between matches involving particular token classes. This transliterator will need to insert the virama character between transliteration rules ending with consonants in order to create consonant clusters.

Configuring

Here you will parameterize the Graph Transliterator using its "easy reading" format, which uses YAML. It maps to a dictionary containing up to five keys: `tokens`, `rules`, `onmatch_rules` (optional), `whitespace`, and `metadata` (optional).

Token Definitions

Graph Transliterator tokenizes its input before transliterating. The `tokens` section will map the input tokens to their token classes. The main class you will need is one for consonants, so you can use `consonant` as the class. Graph Transliterator also requires a dedicated whitespace class, so you can use `whitespace`.

Graph Transliterator allows the use of Unicode character names in files using \N{UNICODE CHARACTER NAME HERE}} notation. You can enter the Unicode characters using that notation or directly. YAML will also unescape \u####, where #### is the hexadecimal notation for a character.

Here is a subsection of that definition:

```
tokens:
  k: [consonant]
  kh: [consonant]
  "\N{LATIN SMALL LETTER N WITH DOT ABOVE}": [consonant]
  a: [vowel]
```

(continues on next page)

(continued from previous page)

```
  aa: [vowel]
  A: [vowel]
  ' ': [wb,whitespace]
  "\t": [wb,whitespace]
  .N: [vowel_sign]
```

Transliteration Rule Definitions

The rule definitions in Graph Transliterator in "easy reading" format are also a dictionary where the rules are the key and the production—what should be outputted by the rule—is the value. For this task, you just need to match individual tokens and also any preceding token classes:

```
rules:
  b: \N{DEVANAGARI LETTER B}
  <consonant> A: \N{DEVANAGARI LETTER AA}
  A: \N{DEVANAGARI LETTER AA}
```

These rules will replace "b" with the devanagari equivalent (ब), and "A" with with a full letter आ if it is at a start of a word (following a token of class "wb", for wordbreak) or otherwise with a vowel sign ा if it is not, presumably following a consonant. Graph Transliterator automatically sorts rules by how many tokens are required for them to be matched, and it picks the one with that requires the most tokens. So the "A" following a consonant would be matched before an "A" after any other character. Graph Transliterator will also check for ambiguity in these rules, unless `check_ambiguity` is set to False.

While not necessary for this tutorial, Graph Transliterator can also require matching of specific previous or following tokens and also classes preceding and following those tokens, e.g.

```
k a r (U M g A <wb>): k,a,r_followed_by_U,M,g,A_and_a_wordbreak
s o (n a): s,o_followed_by_n,a
(<wb> p y) aa r: aa,r_preceded_by_a_wordbreak,p,and_y
```

Here is a subsection of the rules:

```
rules:
  "\t": "\t"
  ' ': ' '
  ',': ','
  .D: "\N{DEVANAGARI LETTER DDDHA}"
  <consonant> A: "\N{DEVANAGARI VOWEL SIGN AA}"
  "\N{LATIN SMALL LETTER N WITH DOT ABOVE}": "\N{DEVANAGARI LETTER NGA}"
```

On Match Rule Definitions

You will want to insert the Virama character between consonants so that they will join together in Unicode output. To do so, add an "onmatch_rules" section:

```
onmatch_rules:
  - <consonant> + <consonant>: "\N{DEVANAGARI SIGN VIRAMA}"
```

Unlike the tokens and rules, the *onmatch rules are ordered*. The first rule matched is applied. In YAML, they consist of a list of dictionaries each with a single key and value. The value is the production string to be inserted between matches. The `+` represents that space. So in the input string *kyA*, which would tokenize as `[' ','k','y','A',' ']`, a virama

character would be inserted when *y* is matched, as it is of class "consonant" and the previously matched transliteration rule for "k" ends with a "consonant".

Whitespace Definitions

The final required setup parameter is for whitespace. These include the `default` whitespace token, which is temporarily added before and after the input tokens; the `consolidate` option to replace sequential whitespace characters with a single default whitespace character; and the `token_class` of whitespace tokens:

```
whitespace:
  consolidate: false
  default: ' '
  token_class: whitespace
```

Metadata Definitions

Graph Transliterator also allows metadata to be added to its settings:

```
metadata:
  title: "ITRANS Devanagari to Unicode"
  version: "0.1.0"
```

Creating a Transliterator

Now that the settings are ready, you can create a Graph Transliterator. Since you have been using the "easy reading" format, you can use `GraphTransliterator.from_yaml_file()` to read from a specific file or the `GraphTransliterator.from_yaml()` to read from a YAML string. You read from the loaded contents of an "easy reading" YAML file using `GraphTransliterator.from_dict()`. Graph Transliterator will convert those settings into basic Python types and then return a `GraphTransliterator`:

```
from graphtransliterator import GraphTransliterator
easyreading_yaml = """
tokens:
  k: [consonant]
  kh: [consonant]
  g: [consonant]
  gh: [consonant]
  ~N: [consonant]
  "\N{LATIN SMALL LETTER N WITH DOT ABOVE}": [consonant]
  ch: [consonant]
  chh: [consonant]
  Ch: [consonant]
  j: [consonant]
  jh: [consonant]
  ~n: [consonant]
  T: [consonant]
  Th: [consonant]
  D: [consonant]
  Dh: [consonant]
  N: [consonant]
  t: [consonant]
  th: [consonant]
```

(continues on next page)

(continued from previous page)

```
  d: [consonant]
  dh: [consonant]
  n: [consonant]
  ^n: [consonant]
  p: [consonant]
  ph: [consonant]
  b: [consonant]
  bh: [consonant]
  m: [consonant]
  y: [consonant]
  r: [consonant]
  R: [consonant]
  l: [consonant]
  ld: [consonant]
  L: [consonant]
  zh: [consonant]
  v: [consonant]
  sh: [consonant]
  Sh: [consonant]
  s: [consonant]
  h: [consonant]
  x: [consonant]
  kSh: [consonant]
  GY: [consonant]
  j~n: [consonant]
  dny: [consonant]
  q: [consonant]
  K: [consonant]
  G: [consonant]
  J: [consonant]
  z: [consonant]
  .D: [consonant]
  .Dh: [consonant]
  f: [consonant]
  Y: [consonant]
  a: [vowel]
  aa: [vowel]
  A: [vowel]
  i: [vowel]
  ii: [vowel]
  I: [vowel]
  ee: [vowel]
  u: [vowel]
  uu: [vowel]
  U: [vowel]
  RRi: [vowel]
  R^i: [vowel]
  LLi: [vowel]
  L^i: [vowel]
  RRI: [vowel]
  LLI: [vowel]
  a.c: [vowel]
  ^e: [vowel]
  e: [vowel]
  ai: [vowel]
  A.c: [vowel]
  ^o: [vowel]
```

(continues on next page)

(continued from previous page)

```
  o: [vowel]
  au: [vowel]
  ' ': [wb,whitespace]
  "\t": [wb,whitespace]
  ',': [wb]
  .h: [wb]
  H: [wb]
  OM: [wb]
  AUM: [wb]
  '|': [wb]
  '||': [wb]
  '0': [wb]
  '1': [wb]
  '2': [wb]
  '3': [wb]
  '4': [wb]
  '5': [wb]
  '6': [wb]
  '7': [wb]
  '8': [wb]
  '9': [wb]
  Rs.: [wb]
  ~Rs.: [wb]
  .a: [wb]
  a.e: [vowel_sign]
  .N: [vowel_sign]
  .n: [vowel_sign]
  M: [vowel_sign]
  .m: [vowel_sign]
rules:
  "\t": "\t"
  ' ': ' '
  ',': ','
  .D: "\N{DEVANAGARI LETTER DDDHA}"
  .Dh: "\N{DEVANAGARI LETTER RHA}"
  .N: "\N{DEVANAGARI SIGN CANDRABINDU}"
  .a: "\N{DEVANAGARI SIGN AVAGRAHA}"
  .h: "\N{DEVANAGARI SIGN VIRAMA}\N{ZERO WIDTH NON-JOINER}"
  .m: "\N{DEVANAGARI SIGN ANUSVARA}"
  .n: "\N{DEVANAGARI SIGN ANUSVARA}"
  '0': "\N{DEVANAGARI DIGIT ZERO}"
  '1': "\N{DEVANAGARI DIGIT ONE}"
  '2': "\N{DEVANAGARI DIGIT TWO}"
  '3': "\N{DEVANAGARI DIGIT THREE}"
  '4': "\N{DEVANAGARI DIGIT FOUR}"
  '5': "\N{DEVANAGARI DIGIT FIVE}"
  '6': "\N{DEVANAGARI DIGIT SIX}"
  '7': "\N{DEVANAGARI DIGIT SEVEN}"
  '8': "\N{DEVANAGARI DIGIT EIGHT}"
  '9': "\N{DEVANAGARI DIGIT NINE}"
  <consonant> A: "\N{DEVANAGARI VOWEL SIGN AA}"
  <consonant> A.c: "\N{DEVANAGARI VOWEL SIGN CANDRA O}"
  <consonant> I: "\N{DEVANAGARI VOWEL SIGN II}"
  <consonant> LLI: "\N{DEVANAGARI VOWEL SIGN VOCALIC LL}"
  <consonant> LLi: "\N{DEVANAGARI VOWEL SIGN VOCALIC L}"
  <consonant> L^i: "\N{DEVANAGARI VOWEL SIGN VOCALIC L}"
  <consonant> RRI: "\N{DEVANAGARI VOWEL SIGN VOCALIC RR}"
```

(continues on next page)

(continued from previous page)

```
  <consonant> RRi: "\N{DEVANAGARI VOWEL SIGN VOCALIC R}"
  <consonant> R^i: "\N{DEVANAGARI VOWEL SIGN VOCALIC R}"
  <consonant> U: "\N{DEVANAGARI VOWEL SIGN UU}"
  <consonant> ^e: "\N{DEVANAGARI VOWEL SIGN SHORT E}"
  <consonant> ^o: "\N{DEVANAGARI VOWEL SIGN SHORT O}"
  <consonant> a: ''
  <consonant> a.c: "\N{DEVANAGARI VOWEL SIGN CANDRA E}"
  <consonant> aa: "\N{DEVANAGARI VOWEL SIGN AA}"
  <consonant> ai: "\N{DEVANAGARI VOWEL SIGN AI}"
  <consonant> au: "\N{DEVANAGARI VOWEL SIGN AU}"
  <consonant> e: "\N{DEVANAGARI VOWEL SIGN E}"
  <consonant> ee: "\N{DEVANAGARI VOWEL SIGN II}"
  <consonant> i: "\N{DEVANAGARI VOWEL SIGN I}"
  <consonant> ii: "\N{DEVANAGARI VOWEL SIGN II}"
  <consonant> o: "\N{DEVANAGARI VOWEL SIGN O}"
  <consonant> u: "\N{DEVANAGARI VOWEL SIGN U}"
  <consonant> uu: "\N{DEVANAGARI VOWEL SIGN UU}"
  A: "\N{DEVANAGARI LETTER AA}"
  A.c: "\N{DEVANAGARI LETTER CANDRA O}"
  AUM: "\N{DEVANAGARI OM}"
  Ch: "\N{DEVANAGARI LETTER CHA}"
  D: "\N{DEVANAGARI LETTER DDA}"
  Dh: "\N{DEVANAGARI LETTER DDHA}"
  G: "\N{DEVANAGARI LETTER GHHA}"
  GY: "\N{DEVANAGARI LETTER JA}\N{DEVANAGARI SIGN VIRAMA}\N{DEVANAGARI LETTER NYA}"
  H: "\N{DEVANAGARI SIGN VISARGA}"
  I: "\N{DEVANAGARI LETTER II}"
  J: "\N{DEVANAGARI LETTER ZA}"
  K: "\N{DEVANAGARI LETTER KHHA}"
  L: "\N{DEVANAGARI LETTER LLA}"
  LLI: "\N{DEVANAGARI LETTER VOCALIC LL}"
  LLi: "\N{DEVANAGARI LETTER VOCALIC L}"
  L^i: "\N{DEVANAGARI LETTER VOCALIC L}"
  M: "\N{DEVANAGARI SIGN ANUSVARA}"
  N: "\N{DEVANAGARI LETTER NNA}"
  OM: "\N{DEVANAGARI OM}"
  R: "\N{DEVANAGARI LETTER RRA}"
  RRI: "\N{DEVANAGARI LETTER VOCALIC RR}"
  RRi: "\N{DEVANAGARI LETTER VOCALIC R}"
  R^i: "\N{DEVANAGARI LETTER VOCALIC R}"
  Rs.: "\N{INDIAN RUPEE SIGN}"
  Sh: "\N{DEVANAGARI LETTER SSA}"
  T: "\N{DEVANAGARI LETTER TTA}"
  Th: "\N{DEVANAGARI LETTER TTHA}"
  U: "\N{DEVANAGARI LETTER UU}"
  Y: "\N{DEVANAGARI LETTER YYA}"
  ^e: "\N{DEVANAGARI LETTER SHORT E}"
  ^n: "\N{DEVANAGARI LETTER NNNA}"
  ^o: "\N{DEVANAGARI LETTER SHORT O}"
  a: "\N{DEVANAGARI LETTER A}"
  a.c: "\N{DEVANAGARI LETTER CANDRA E}"
  a.e: "\N{DEVANAGARI LETTER CANDRA A}"
  aa: "\N{DEVANAGARI LETTER AA}"
  ai: "\N{DEVANAGARI LETTER AI}"
  au: "\N{DEVANAGARI LETTER AU}"
  b: "\N{DEVANAGARI LETTER BA}"
  bh: "\N{DEVANAGARI LETTER BHA}"
```

(continues on next page)

(continued from previous page)

```
  ch: "\N{DEVANAGARI LETTER CA}"
  chh: "\N{DEVANAGARI LETTER CHA}"
  d: "\N{DEVANAGARI LETTER DA}"
  dh: "\N{DEVANAGARI LETTER DHA}"
  dny: "\N{DEVANAGARI LETTER JA}\N{DEVANAGARI SIGN VIRAMA}\N{DEVANAGARI LETTER NYA}"
  e: "\N{DEVANAGARI LETTER E}"
  ee: "\N{DEVANAGARI LETTER II}"
  f: "\N{DEVANAGARI LETTER FA}"
  g: "\N{DEVANAGARI LETTER GA}"
  gh: "\N{DEVANAGARI LETTER GHA}"
  h: "\N{DEVANAGARI LETTER HA}"
  i: "\N{DEVANAGARI LETTER I}"
  ii: "\N{DEVANAGARI LETTER II}"
  j: "\N{DEVANAGARI LETTER JA}"
  jh: "\N{DEVANAGARI LETTER JHA}"
  j~n: "\N{DEVANAGARI LETTER JA}\N{DEVANAGARI SIGN VIRAMA}\N{DEVANAGARI LETTER NYA}"
  k: "\N{DEVANAGARI LETTER KA}"
  kSh: "\N{DEVANAGARI LETTER KA}\N{DEVANAGARI SIGN VIRAMA}\N{DEVANAGARI LETTER SSA}"
  kh: "\N{DEVANAGARI LETTER KHA}"
  l: "\N{DEVANAGARI LETTER LA}"
  ld: "\N{DEVANAGARI LETTER LLA}"
  m: "\N{DEVANAGARI LETTER MA}"
  n: "\N{DEVANAGARI LETTER NA}"
  o: "\N{DEVANAGARI LETTER O}"
  p: "\N{DEVANAGARI LETTER PA}"
  ph: "\N{DEVANAGARI LETTER PHA}"
  q: "\N{DEVANAGARI LETTER QA}"
  r: "\N{DEVANAGARI LETTER RA}"
  s: "\N{DEVANAGARI LETTER SA}"
  sh: "\N{DEVANAGARI LETTER SHA}"
  t: "\N{DEVANAGARI LETTER TA}"
  th: "\N{DEVANAGARI LETTER THA}"
  u: "\N{DEVANAGARI LETTER U}"
  uu: "\N{DEVANAGARI LETTER UU}"
  v: "\N{DEVANAGARI LETTER VA}"
  x: "\N{DEVANAGARI LETTER KA}\N{DEVANAGARI SIGN VIRAMA}\N{DEVANAGARI LETTER SSA}"
  y: "\N{DEVANAGARI LETTER YA}"
  z: "\N{DEVANAGARI LETTER ZA}"
  zh: "\N{DEVANAGARI LETTER LLLA}"
  '|': "\N{DEVANAGARI DANDA}"
  '||': "\N{DEVANAGARI DOUBLE DANDA}"
  ~N: "\N{DEVANAGARI LETTER NGA}"
  ~Rs.: "\N{INDIAN RUPEE SIGN}"
  ~n: "\N{DEVANAGARI LETTER NYA}"
  "\N{LATIN SMALL LETTER N WITH DOT ABOVE}": "\N{DEVANAGARI LETTER NGA}"
onmatch_rules:
- <consonant> + <consonant>: "\N{DEVANAGARI SIGN VIRAMA}"
whitespace:
  consolidate: false
  default: ' '
  token_class: whitespace
metadata:
  title: ITRANS to Unicode
  version: 0.1.0
"""
gt = GraphTransliterator.from_yaml(easyreading_yaml)
```

Transliterating

With the transliterator created, you can now transliterate using `GraphTransliterator.transliterate()`:

```
250 gt.transliterate("aaj mausam ba.Daa beiimaan hai, aaj mausam")
```

```
'□□ □□□□ □□□ □□□□□□ □□, □□ □□□□'
```

Other Information

Graph Transliterator has a few other tools built in that are for more specialized applications.

If you want to receive the details of the most recent transliteration, access `GraphTransliterator.last_matched_rules` to get this list of rules matched:

```
251 gt.last_matched_rules
```

```
[TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['aa
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['j
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['m
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['au'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['s
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['m
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['b
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['.D
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['aa'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['b
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['e'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['ii
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['m
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='□', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['aa'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='□', prev_classes=None, prev_tokens=None, tokens=['n
```

(continues on next page)

(continued from previous page)

```
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['h
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ै', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['ai'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production=',', prev_classes=None, prev_tokens=None, tokens=[',
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['aa
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['j
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=['
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['m
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='ौ', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['au'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['s
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562),
 TransliterationRule(production='', prev_classes=['consonant'], prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.41503749927884376),
 TransliterationRule(production='ा', prev_classes=None, prev_tokens=None, tokens=['m
↪'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

Or if you just want to know the tokens matched by each rule, check `GraphTransliterator.last_matched_rule_tokens`:

```
gt.last_matched_rule_tokens
```

```
[['aa'],
 ['j'],
 [' '],
 ['m'],
 ['au'],
 ['s'],
 ['a'],
 ['m'],
 [' '],
 ['b'],
 ['a'],
 ['.D'],
 ['aa'],
 [' '],
 ['b'],
 ['e'],
 ['ii'],
 ['m'],
 ['aa'],
 ['n'],
 [' '],
 ['h'],
 ['ai'],
```

(continues on next page)

(continued from previous page)

```
 [','],
 [' '],
 ['aa'],
 ['j'],
 [' '],
 ['m'],
 ['au'],
 ['s'],
 ['a'],
 ['m']]
```

You can access the directed tree used by GraphTransliterator using `GraphTransliterator.graph`:

```
gt.graph
```

```
<graphtransliterator.graphs.DirectedGraph at 0x7f8aac66be40>
```

3.2.6 Advanced Tutorial: Bundling a Transliterator

This advanced tutorial builds upon the original tutorial to show you how to bundle a transliterator for inclusion in Graph Transliterator.

Contributions to Graph Transliterator are strongly encouraged!

You will make a very simple transliterator while going through the steps of bundling it into Graph Transliterator.

Git Basics: Fork, Branch, Sync, Commit

Fork

The first thing to do, if you have not already, is to create a fork of Graph Transliterator. See https://help.github.com/en/articles/fork-a-repo

(From here on out, we will be using the command line.)

After creating a fork, clone your forked repo:

```
git clone https://github.com/YOUR-USERNAME/graphtransliterator
```

Branch

Once you have done that, go into that directory and create a new branch:

```
cd graphtransliterator
git checkout -b [name_of_your_transliterator_branch]
```

For this example, you can use the branch *a_to_b*:

```
cd graphtransliterator
git checkout -b a_to_b
```

Then, push that branch to the origin (your personal github fork):

```
git push origin [name_of_your_transliterator_branch]

Here that would be:

.. code-block:: bash

  git push origin a_to_b
```

Next, add a remote upstream for Graph Transliterator (the official Graph Transliterator repo):

```
git remote add upstream https://github.com/seanpue/graphtransliterator.git
```

Sync

To update your local copy of the the remote (official Graph Transliterator repo), run:

```
git fetch upstream
```

To sync your personal fork with the remote, run:

```
git merge upstream/master
```

See https://help.github.com/en/articles/syncing-a-fork for more info. You can run the previous two commands at any time.

Commit

You can commit your changes by running:

```
git commit -m 'comment here about the commit'
```

Adding A Transliterator

To add a transliterator, the next step is to create a subdirectory in `transliterators`. For this tutorial, you can make a branch named `a_to_b`.

Note that this will be under `graphtransliterator/transliterators`, so from the root directory enter:

```
cd graphtransliterator/transliterators
mkdir [name_of_your_transliterator]
cd [name_of_your_transliterator]
```

For this example, you would enter:

```
cd graphtransliterator/transliterators
mkdir a_to_b
cd a_to_b
```

In the `graphtransliterator/transliterators/[name_of_your_transliterator]` directory, you will add:

- an *__init__.py*
- a YAML file in the "easy reading format"

- a JSON file that is a serialization of the transliterator (optional)
- a `tests` directory including a file named `[name_of_your_transliterator]_tests.yaml`
- a Python test named `test_[name_of_your_transliterator].py` (optional)

Here is a tree showing the file organization:

```
transliterators
├── {{source_to_target}}
|   ├── __init__.py
|   ├── {{source_to_target}}.json
|   ├── {{source_to_target}}.yaml
└── tests
    ├── test_{{source_to_target}}.py
    └── {{source_to_target}}_tests.yaml
```

YAML File

The YAML file should contain the "easy reading" version of your transliterator. For this example, create a file called `a_to_b.yaml`. Add a `metadata` field to the YAML file, as well, following the guidelines.

```yaml
tokens:
  a: [a_class]
  ' ': [whitespace]
rules:
  a: A
onmatch_rules:
  - <a_class> + <a_class>: ","
whitespace:
  default: ' '
  token_class: whitespace
  consolidate: false
metadata:
  name: A to B
  version: 0.0.1
  url: http://website_of_project.com
  author: Your Name is Optional
  author_email: your_email@is_option.al
  maintainer: Maintainer's Name is Optional
  maintainer_email: maintainers_email@is_option.al
  license: MIT or Other Open Source License
  keywords: [add, keywords, here, as, a, list]
  project_urls:
     Documentation: https://link_to_documentation.html
     Source: https://link_to_sourcecode.html
     Tracker: https://link_to_issue_tracker.html
```

For most use cases, the `project_urls` can link to the Graph Transliterator Github page.

JSON File

To create a JSON file, you can use the command line interface:

$ graphtransliterator dump –from yaml_file a_to_b.yaml > a_to_b.json

Alternatively, you can use the `make-json` command:

$ graphtransliterator make-json AToB

The JSON file loads more quickly than the YAML one, but it is not necessary during development.

__init__.py

The *__init__.py* will create the bundled transliterator, which is a subclass of *GraphTransliterator* named *Bundled*.

Following convention, uou need to name your transliterator's class is CamelCase. For this example, it would be `AToB`:

```
from graphtransliterator.transliterators import Bundled

class AToB(Bundled):
    """
    A to B Bundled Graph Transliterator
    """

    def __init__(self, **kwargs):
        """Initialize transliterator from YAML."""
        self.from_YAML(
            **kwargs
        )  # defaults to check_ambiguity=True, check_coverage=True
        # When ready, remove the previous lines and initialize more quickly from JSON:
        # self.init_from_JSON(**kwargs) # check_ambiguity=False, check_coverage=False
```

When you load the bundled transliterator from YAML using `from_YAML` it will check for ambiguity as well as check the coverage of the tests. You can turn those features off temporarily here.

When a transliterator is added into Graph Transliterator, it will likely be set to load from JSON by default. Tests will check for ambiguity and coverage.

Tests

Graph Transliterator requires that all bundled transliterators have tests that visit every edge and node of the internal graph and that use all on-match rules. The test file should be a YAML file defining a dictionary keyed from input to correct output.

You can test the transliterator as you are developing it by adding YAML tests and running the command:

```
graphtransliterator test [name_of_your_transliterator]
```

Tests can be generated using the command line interface:

```
mkdir tests
graphtransliterator generate-tests --from bundled [name_of_your_transliterator] >␣
↪tests/[name_of_your_transliterator]
```

Testing the Transliterator

You should test the transliterator to make sure everything is correct, including its metadata. To do that, navigate back to the root directory of *graphtransliterator* and execute the command:

```
py.test tests/test_transliterators.py
```

You can also run the complete suite of tests by running:

```
tox
```

Pushing Your Transliterator

When you are finished with a version of your transliterator, you should once again commit it to your github branch after syncing your branch with the remote. Then you can make a pull request to include the transliterator in Graph Transliterator. You can do that from the Graph Transliterator Github page. See https://help.github.com/en/articles/creating-a-pull-request-from-a-fork.

3.2.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Contributor Code of Conduct

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

Types of Contributions

You can contribute in many ways:

Report Bugs

Report bugs at https://github.com/seanpue/graphtransliterator/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

Graph-based Transliterator could always use more documentation, whether as part of the official Graph-based Transliterator docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/seanpue/graphtransliterator/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Add Transliterators

We welcome new transliterators to be added to the bundled transliterators!

See the documentation about Bundled Transliterators and look at Example as a model.

Raise an issue on Github, https://github.com/seanpue/graphtransliterator/issues

Then create a new branch with the new transliterator. Make sure the transliterator passes all of these requirements:

- is a submodule of graphtransliterator.transliterators
- has a unique name, preferably in format source_to_target
- has the following files: - __init__.py - {{source_to_target}}.yaml - {{source_to_target}}.json - tests/{{source_to_target}}_tests.yaml - tests/test_{{source_to_target}}.py (optional)
- has a classname in camel case, e.g. SourceToTarget
- has complete test coverage of all nodes and edges of generated graph and all onmatch rules, if present
- has required metadata in the YAML file.

When all the requirements are fulfilled, submit a pull request, and it will be reviewed for inclusion in a near-future release.

Get Started!

Ready to contribute? Here's how to set up *graphtransliterator* for local development.

1. Fork the *graphtransliterator* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/graphtransliterator.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv graphtransliterator
$ cd graphtransliterator/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, format your code using the Black code formatter. (You can do that in your editor, as well). Then check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ black graphtransliterator
$ flake8 graphtransliterator tests
$ python setup.py test or py.test
$ tox
```

   To get black, flake8, and tox, just pip install them into your virtualenv.

   You should also test your coverage using make:

   $ make coverage

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7 and 3.8 for PyPy. Check https://travis-ci.org/seanpue/graphtransliterator/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_graphtransliterator
```

Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

The module uses Github Actions to deploy to TestPyPI and to PyPI.

3.2.8 API Reference

A list of the full API reference of all public classes and functions is below.

Public members can (and should) be imported from *graphtransliterator*:

```
from graphtransliterator import GraphTransliterator
```

Bundled transliterators require that *graphtransliterator.transliterators*: be imported:

```
import graphtransliterator.transliterators
transliterators.iter_names()
```

Core Classes

class graphtransliterator.**GraphTransliterator**(*tokens*, *rules*, *whitespace*, *onmatch_rules=None*, *metadata=None*, *ignore_errors=False*, *check_ambiguity=True*, *onmatch_rules_lookup=None*, *tokens_by_class=None*, *graph=None*, *tokenizer_pattern=None*, *graphtransliterator_version=None*, ***kwargs*)

A graph-based transliteration tool that lets you convert the symbols of one language or script to those of another using rules that you define.

Transliteration of tokens of an input string to an output string is configured by: a set of input token types with classes, pattern-matching rules involving sequences of tokens as well as preceding or following tokens and token classes, insertion rules between matches, and optional consolidation of whitespace. Rules are ordered by specificity.

Note: This constructor does not validate settings and should typically not be called directly. Use *from_dict()* instead. For "easy reading" support, use *from_easyreading_dict()*, *from_yaml()*, or *from_yaml_file()*. Keyword parameters used here (`ignore_errors`, `check_ambiguity`) can be passed from those other constructors.

Parameters

- **tokens** (*dict* of {*str*: *set* of *str*}) – Mapping of input token types to token classes
- **rules** (*list* of *TransliterationRule*) – *list* of transliteration rules ordered by cost
- **onmatch_rules** (*list* of `OnMatchRule`, or *None*) – Rules for output to be inserted between tokens of certain classes when a transliteration rule has been matched but before its production string has been added to the output
- **whitespace** (*WhitespaceRules*) – Rules for handling whitespace
- **metadata** (*dict* or *None*) – Metadata settings
- **ignore_errors** (*bool*, optional) – If true, transliteration errors are ignored and do not raise an exception. The default is false.
- **check_ambiguity** (*bool*, optional) – If true (default), transliteration rules are checked for ambiguity. `load()` and `loads()` do not check ambiguity by default.
- **onmatch_rules_lookup** (*dict* of {*str*: dict of {*str*: *list* of *int*}}, optional`) – OnMatchRules lookup, used internally, will be generated if not present.
- **tokens_by_class** (*dict* of {*str*: *set* of *str*}, optional) – Tokens by class, used internally, will be generated if not present.
- **graph** (*DirectedGraph*, optional) – Directed graph used by Graph Transliterator, will be generated if not present.
- **tokenizer_pattern** (*str*, optional) – Regular expression pattern for input string tokenization, will be generated if not present.
- **graphtransliterator_version** (*str*, optional) – Version of graphtransliterator, added by *dump()* and *dumps()*.

Example

```
from graphtransliterator import GraphTransliterator, OnMatchRule, TransliterationRule, WhitespaceRules
settings = {'tokens': {'a': {'vowel'}, ' ': {'wb'}}, 'onmatch_rules':[OnMatchRule(prev_classes=['vowel'], next_classes=['vowel'], production=',')], 'rules': [TransliterationRule(production='A', prev_classes=None, prev_tokens=None, tokens=['a'], next_tokens=None, next_classes=None, cost=0.5849625007211562), TransliterationRule(production=' ', prev_classes=None, prev_tokens=None, tokens=[' '], next_tokens=None, next_classes=None, cost=0.5849625007211562)], 'metadata': {'author': 'Author McAuthorson'}, 'whitespace':WhitespaceRules(default=' ', token_class='wb', consolidate=False)}
gt = GraphTransliterator(**settings)
gt.transliterate('a')
```

```
'A'
```

See also:

from_dict
Constructor from dictionary of settings

from_easyreading_dict
Constructor from dictionary in “easy reading” format

from_yaml
Constructor from YAML string in “easy reading” format

from_yaml_file
Constructor from YAML file in “easy reading” format

dump (*compression_level=0*)

Dump configuration of Graph Transliterator to Python data types.

Compression is turned off by default.

Parameters
compression_level (*int*) – A value in 0 (default, no compression), 1 (compression including graph), and 2 (compressiong without graph)

Returns

GraphTransliterator configuration as a dictionary with keys:

"tokens"
Mappings of tokens to their classes (*OrderedDict* of {str: *list* of *str*})

"rules"
Transliteration rules in direct format (*list* of *dict* of {*str*: *str*})

"whitespace"
Whitespace settings (*dict* of {*str*: *str*})

"onmatch_rules"
On match rules (*list* of *OrderedDict*)

"metadata"
Dictionary of metadata (*dict*)

"ignore_errors"
Ignore errors in transliteration (*bool*)

"onmatch_rules_lookup"
Dictionary keyed by current token to previous token containing a list of indexes of applicable `OnmatchRule` to try (*dict* of {*str*: *dict* of {*str*: *list* of *int*}})

"tokens_by_class"
Tokens keyed by token class, used internally (*dict* of {*str*: *list* of str})

"graph"
Serialization of *DirectedGraph* (*dict*)

"tokenizer_pattern"
Regular expression for tokenizing (*str*)

"graphtransliterator_version"
Module version of *graphtransliterator* (*str*)

Return type
OrderedDict

Example

```
yaml_ = '''
tokens:
  a: [vowel]
  ' ': [wb]
rules:
  a: A
  ' ': ' '
whitespace:
  default: " "
  consolidate: false
  token_class: wb
onmatch_rules:
  - <vowel> + <vowel>: ','  # add a comma between vowels
metadata:
  author: "Author McAuthorson"
'''
gt = GraphTransliterator.from_yaml(yaml_)
gt.dump()
```

```
OrderedDict([('tokens', {'a': ['vowel'], ' ': ['wb']}),
             ('rules',
              [OrderedDict([('production', 'A'),
                            ('tokens', ['a']),
                            ('cost', 0.5849625007211562)]),
               OrderedDict([('production', ' '),
                            ('tokens', [' ']),
                            ('cost', 0.5849625007211562)])]),
             ('whitespace',
              {'default': ' ', 'token_class': 'wb', 'consolidate': False}),
             ('onmatch_rules',
              [OrderedDict([('prev_classes', ['vowel']),
                            ('next_classes', ['vowel']),
                            ('production', ',')])]),
             ('metadata', {'author': 'Author McAuthorson'}),
             ('ignore_errors', False),
             ('onmatch_rules_lookup', {'a': {'a': [0]}}),
             ('tokens_by_class', {'vowel': ['a'], 'wb': [' ']}),
             ('graph',
              {'node': [{'type': 'Start',
                 'ordered_children': {'a': [1], ' ': [3]}},
                {'token': 'a',
                 'type': 'token',
                 'ordered_children': {'__rules__': [2]}},
                {'type': 'rule', 'accepting': True, 'rule_key': 0},
                {'token': ' ',
                 'type': 'token',
                 'ordered_children': {'__rules__': [4]}},
                {'type': 'rule', 'accepting': True, 'rule_key': 1}],
               'edge': {0: {1: {'token': 'a', 'cost': 0.5849625007211562},
                 3: {'token': ' ', 'cost': 0.5849625007211562}},
                1: {2: {'cost': 0.5849625007211562}},
                3: {4: {'cost': 0.5849625007211562}}},
               'edge_list': [(0, 1), (0, 3), (1, 2), (3, 4)]}),
             ('tokenizer_pattern', '(a|\\ )'),
             ('graphtransliterator_version', '1.2.4')])
```

See also:

dumps
Dump Graph Transliterator configuration to JSON string

load
Load Graph Transliteration from configuration in Python data types

loads
Load Graph Transliteration from configuration as a JSON string

dumps (*compression_level=2*)

Parameters

- **compression_level** (*int*) – A value in 0 (no compression), 1 (compression including graph), and 2 (default, compression without graph)
- **separators** (*tuple* of *str*) – Separators used by json.dumps(), default is compact
- **(JSON).** (*Dump settings of Graph Transliterator to Javascript Object Notation*) –
- **default.** (*Compression is turned on by*) –

Returns
JSON string

Return type
str

Examples

```
yaml_ = '''
  tokens:
    a: [vowel]
    ' ': [wb]
  rules:
    a: A
    ' ': ' '
  whitespace:
    default: " "
    consolidate: false
    token_class: wb
  onmatch_rules:
    - <vowel> + <vowel>: ','  # add a comma between vowels
  metadata:
    author: "Author McAuthorson"
'''
gt = GraphTransliterator.from_yaml(yaml_)
gt.dumps()
```

```
'{"graphtransliterator_version":"1.2.4","compressed_settings":[["vowel","wb"],
↪[" ","a"],[[1],[0]],[["A",0,0,[1],0,0,-1],[" ",0,0,[0],0,0,-1]],[" ","wb",
↪0],[[[0],[0],","]],{"author":"Author McAuthorson"},null]}'
```

See also:

dump
Dump Graph Transliterator configuration to Python data types

load
Load Graph Transliteration from configuration in Python data types

loads
Load Graph Transliteration from configuration as a JSON string

static from_dict (*dict_settings*, ***kwargs*)

Generate GraphTransliterator from *dict* settings.

Parameters
dict_settings (*dict*) – Dictionary of settings

Returns
Graph transliterator

Return type
GraphTransliterator

static from_easyreading_dict (*easyreading_settings*, ***kwargs*)

Constructs *GraphTransliterator* from a dictionary of settings in "easy reading" format, i.e. the loaded contents of a YAML string.

Parameters
easyreading_settings (*dict*) – Settings dictionary in easy reading format with keys:

"tokens"
Mappings of tokens to their classes (*dict* of {str: *list* of *str*})

"rules"
Transliteration rules in "easy reading" format (*list* of *dict* of {*str*: *str*})

"onmatch_rules"
On match rules in "easy reading" format (*dict* of {*str*: *str*}, optional)

"whitespace"
Whitespace definitions, including default whitespace token, class of whitespace tokens, and whether or not to consolidate (*dict* of {'default': *str*, 'token_class': *str*, consolidate: *bool*}, optional)

"metadata"
Dictionary of metadata (*dict*, optional)

Returns
Graph Transliterator

Return type
GraphTransliterator

Note: Called by `from_yaml()`.

Example

```
tokens = {
    'ab': ['class_ab'],
    ' ': ['wb']
}
whitespace = {
    'default': ' ',
    'token_class': 'wb',
    'consolidate': True
}
onmatch_rules = [
    {'<class_ab> + <class_ab>': ','}
]
rules = {'ab': 'AB',
         ' ': '_'}
settings = {'tokens': tokens,
            'rules': rules,
            'whitespace': whitespace,
            'onmatch_rules': onmatch_rules}
gt = GraphTransliterator.from_easyreading_dict(settings)
gt.transliterate("ab abab")
```

```
'AB_AB,AB'
```

See also:

from_yaml
Constructor from YAML string in "easy reading" format

from_yaml_file
Constructor from YAML file in "easy reading" format

static from_yaml (*yaml_str*, *charnames_escaped=True*, ***kwargs*)

Construct GraphTransliterator from a YAML str.

Parameters

- **yaml_str** (*str*) – YAML mappings of tokens, rules, and (optionally) onmatch_rules
- **charnames_escaped** (*boolean*) – Unescape Unicode during YAML read (default True)

Note: Called by *from_yaml_file()* and calls *from_easyreading_dict()*.

Example

```
yaml_ = '''
tokens:
  a: [class1]
  ' ': [wb]
rules:
  a: A
  ' ': ' '
whitespace:
  default: ' '
  consolidate: True
  token_class: wb
onmatch_rules:
  - <class1> + <class1>: "+"
'''
gt = GraphTransliterator.from_yaml(yaml_)
gt.transliterate("a aa")
```

```
'A A+A'
```

See also:

from_easyreading_dict
Constructor from dictionary in “easy reading” format

from_yaml
Constructor from YAML string in “easy reading” format

from_yaml_file
Constructor from YAML file in “easy reading” format

static from_yaml_file(*yaml_filename*, ***kwargs*)

Construct GraphTransliterator from YAML file.

Parameters
yaml_filename (*str*) – Name of YAML file, containing tokens, rules, and (optionally) onmatch_rules

---

Note: Calls *from_yaml()*.

See also:

from_yaml
Constructor from YAML string in “easy reading” format

from_easyreading_dict
Constructor from dictionary in “easy reading” format

property graph

Graph used in transliteration.

Type
DirectedGraph

property graphtransliterator_version

Graph Transliterator version.

Type
str

property ignore_errors

Ignore transliteration errors setting.

Type
bool

property last_input_tokens

Last tokenization of the input string, with whitespace at start and end.

Type
list of *str*

property last_matched_rule_tokens

Last matched tokens for each rule.

Type
list of *list* of *str*

property last_matched_rules

Last transliteration rules matched.

Type
list of *TransliterationRule*

static load(*settings*, ***kwargs*)

Create GraphTransliterator from settings as Python data types.

Parameters
settings – GraphTransliterator configuration as a dictionary with keys:

"tokens"
Mappings of tokens to their classes (*dict* of {str: *list* of *str*})

"rules"
Transliteration rules in direct format (*list* of *OrderedDict* of {*str*: *str*})

"whitespace"
Whitespace settings (*dict* of {*str*: *str*})

"onmatch_rules"
On match rules (*list* of *OrderedDict*, optional)

"metadata"
Dictionary of metadata (*dict*, optional)

"ignore_errors"
Ignore errors. (*bool*, optional)

"onmatch_rules_lookup"
Dictionary keyed by current token to previous token containing a list of indexes of applicable `OnmatchRule` to try (*dict* of {*str*: *dict* of {*str*: *list* of *int*}}, optional)

tokens_by_class
Tokens keyed by token class, used internally (*dict* of {*str*: *list* of str}, optional)

graph
Serialization of *DirectedGraph* (*dict*, optional)

"tokenizer_pattern"
Regular expression for tokenizing (*str*, optional)

"graphtransliterator_version"
Module version of *graphtransliterator* (*str*, optional)

Returns
Graph Transliterator

Return type
GraphTransliterator

Example

```
from collections import OrderedDict
settings =           {'tokens': {'a': ['vowel'], ' ': ['wb']},
 'rules': [OrderedDict([('production', 'A'),
                # Can be compacted, removing None values
                # ('prev_tokens', None),
                ('tokens', ['a']),
                ('next_classes', None),
                ('next_tokens', None),
                ('cost', 0.5849625007211562)]),
  OrderedDict([('production', ' '),
                ('prev_classes', None),
                ('prev_tokens', None),
                ('tokens', [' ']),
                ('next_classes', None),
                ('next_tokens', None),
                ('cost', 0.5849625007211562)])],
 'whitespace': {'default': ' ', 'token_class': 'wb', 'consolidate': False},
 'onmatch_rules': [OrderedDict([('prev_classes', ['vowel']),
                ('next_classes', ['vowel']),
                ('production', ',')])],
 'metadata': {'author': 'Author McAuthorson'},
 'onmatch_rules_lookup': {'a': {'a': [0]}},
 'tokens_by_class': {'vowel': ['a'], 'wb': [' ']},
 'graph': {'edge': {0: {1: {'token': 'a', 'cost': 0.5849625007211562},
    3: {'token': ' ', 'cost': 0.5849625007211562}},
   1: {2: {'cost': 0.5849625007211562}},
   3: {4: {'cost': 0.5849625007211562}}},
  'node': [{'type': 'Start', 'ordered_children': {'a': [1], ' ': [3]}},
   {'type': 'token', 'token': 'a', 'ordered_children': {'__rules__': [2]}},
   {'type': 'rule',
    'rule_key': 0,
    'accepting': True,
    'ordered_children': {}},
   {'type': 'token', 'token': ' ', 'ordered_children': {'__rules__': [4]}},
   {'type': 'rule',
    'rule_key': 1,
    'accepting': True,
    'ordered_children': {}}],
  'edge_list': [(0, 1), (1, 2), (0, 3), (3, 4)]},
 'tokenizer_pattern': '(a|\ )',
 'graphtransliterator_version': '0.3.3'}
gt = GraphTransliterator.load(settings)
gt.transliterate('aa')
```

```
'A,A'
```

```
# can be compacted
settings.pop('onmatch_rules_lookup')
GraphTransliterator.load(settings).transliterate('aa')
```

```
'A,A'
```

See also:

dump
Dump Graph Transliterator configuration to Python data types

dumps
Dump Graph Transliterator configuration to JSON string

loads
Load Graph Transliteration from configuration as a JSON string

static loads (*settings*, ***kwargs*)

Create GraphTransliterator from JavaScript Object Notation (JSON) string.

Parameters
settings – JSON settings for GraphTransliterator

Returns
Graph Transliterator

Return type
GraphTransliterator

Example

```
JSON_settings = '''{"tokens": {"a": ["vowel"], " ": ["wb"]}, "rules": [{
↪"production": "A", "prev_classes": null, "prev_tokens": null, "tokens": ["a
↪"], "next_classes": null, "next_tokens": null, "cost": 0.5849625007211562},
↪{"production": " ", "prev_classes": null, "prev_tokens": null, "tokens": ["
↪"], "next_classes": null, "next_tokens": null, "cost": 0.5849625007211562}],
↪ "whitespace": {"default": " ", "token_class": "wb", "consolidate": false},
↪"onmatch_rules": [{"prev_classes": ["vowel"], "next_classes": ["vowel"],
↪"production": ","}], "metadata": {"author": "Author McAuthorson"}, "ignore_
↪errors": false, "onmatch_rules_lookup": {"a": {"a": [0]}}, "tokens_by_class
↪": {"vowel": ["a"], "wb": [" "]}, "graph": {"node": [{"type": "Start",
↪"ordered_children": {"a": [1], " ": [3]}}, {"type": "token", "token": "a",
↪"ordered_children": {"__rules__": [2]}}, {"type": "rule", "rule_key": 0,
↪"accepting": true, "ordered_children": {}}, {"type": "token", "token": " ",
↪"ordered_children": {"__rules__": [4]}}, {"type": "rule", "rule_key": 1,
↪"accepting": true, "ordered_children": {}}], "edge": {"0": {"1": {"token":
↪"a", "cost": 0.5849625007211562}, "3": {"token": " ", "cost": 0.
↪5849625007211562}}, "1": {"2": {"cost": 0.5849625007211562}}, "3": {"4": {
↪"cost": 0.5849625007211562}}}, "edge_list": [[0, 1], [1, 2], [0, 3], [3,␣
↪4]]}, "tokenizer_pattern": "(a| )", "graphtransliterator_version": "1.2.2"}'
↪''

gt = GraphTransliterator.loads(JSON_settings)
gt.transliterate('a')
```

```
'A'
```

See also:

dump
Dump Graph Transliterator configuration to Python data types

dumps
Dump Graph Transliterator configuration to JSON string

load
Load Graph Transliteration from configuration in Python data types

match_at (*token_i*, *tokens*, *match_all=False*)

Match best (least costly) transliteration rule at a given index in the input tokens and return the index to that rule. Optionally, return all rules that match.

Parameters

- **token_i** (*int*) – Location in *tokens* at which to begin
- **tokens** (*list* of *str*) – List of tokens
- **match_all** (*bool*, optional) – If true, return the index of all rules matching at the given index. The default is false.

Returns
Index of matching transliteration rule in *GraphTransliterator.rules* or None. Returns a *list* of *int* or an empty *list* if `match_all` is true.

Return type
int, *None*, or *list* of *int*

Note: Expects whitespaces token at beginning and end of *tokens*.

Examples

```
gt = GraphTransliterator.from_yaml('''
        tokens:
            a: []
            a a: []
            ' ': [wb]
        rules:
            a: <A>
            a a: <AA>
        whitespace:
            default: ' '
            consolidate: True
            token_class: wb
''')
tokens = gt.tokenize("aa")
tokens # whitespace added to ends
```

```
[' ', 'a', 'a', ' ']
```

```
142 gt.match_at(1, tokens) # returns index to rule
```

```
0
```

```
143 gt.rules[gt.match_at(1, tokens)] # actual rule
```

```
TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a', 'a'], next_tokens=None, next_classes=None, cost=0.
↪41503749927884376)
```

```
144 gt.match_at(1, tokens, match_all=True) # index to rules, with match_all
```

```
[0, 1]
```

```
145 [gt.rules[_] for _ in gt.match_at(1, tokens, match_all=True)]
```

```
[TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a', 'a'], next_tokens=None, next_classes=None, cost=0.
↪41503749927884376),
 TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

property metadata

Metadata of transliterator

Type
dict

property onmatch_rules

Rules for productions between matches.

Type
list of `OnMatchRules`

property onmatch_rules_lookup

On Match Rules lookup

Type
dict

property productions

List of productions of each transliteration rule.

Type
list of *str*

pruned_of (*productions*)

Remove transliteration rules with specific output productions.

Parameters
productions (*str*, or *list* of *str*) – list of productions to remove

Returns
Graph transliterator pruned of certain productions.

Return type
graphtransliterator.GraphTransliterator

Note: Uses original initialization parameters to construct a new *GraphTransliterator*.

Examples

```
gt = GraphTransliterator.from_yaml('''
        tokens:
            a: []
            a a: []
            ' ': [wb]
        rules:
            a: <A>
            a a: <AA>
        whitespace:
            default: ' '
            consolidate: True
            token_class: wb
''')
gt.rules
```

```
[TransliterationRule(production='<AA>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a', 'a'], next_tokens=None, next_classes=None, cost=0.
↪41503749927884376),
 TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

```
gt.pruned_of('<AA>').rules
```

```
[TransliterationRule(production='<A>', prev_classes=None, prev_tokens=None,␣
↪tokens=['a'], next_tokens=None, next_classes=None, cost=0.5849625007211562)]
```

```
gt.pruned_of(['<A>', '<AA>']).rules
```

```
[]
```

property rules

Transliteration rules sorted by cost.

Type
list of *TransliterationRule*

tokenize (*input*)

Tokenizes an input string.

Adds initial and trailing whitespace, which can be consolidated.

Parameters
input (*str*) – String to tokenize

Returns
List of tokens, with default whitespace token at beginning and end.

Return type
list of *str*

Raises
ValueError – Unrecognizable input, such as a character that is not in a token

Examples

```
tokens = {'ab': ['class_ab'], ' ': ['wb']}
whitespace = {'default': ' ', 'token_class': 'wb', 'consolidate': True}
rules = {'ab': 'AB', ' ': '_'}
settings = {'tokens': tokens, 'rules': rules, 'whitespace': whitespace}
gt = GraphTransliterator.from_easyreading_dict(settings)
gt.tokenize('ab ')
```

```
[' ', 'ab', ' ']
```

property tokenizer_pattern
Tokenizer pattern from transliterator

Type
str

property tokens
Mappings of tokens to their classes.

Type
dict of {*str*

Type
set of *str*}

property tokens_by_class
Tokenizer pattern from transliterator

Type
dict of {*str*

Type
list of *str*}

transliterate(*input*)
Transliterate an input string into an output string.

Parameters
input (*str*) – Input string to transliterate

Returns
Transliteration output string

Return type
str

Raises
ValueError – Cannot parse input

Note: Whitespace will be temporarily appended to start and end of input string.

Example

```
GraphTransliterator.from_yaml(
'''
tokens:
  a: []
  ' ': [wb]
rules:
  a: A
  ' ': '_'
whitespace:
  default: ' '
  consolidate: True
  token_class: wb
''').transliterate("a a")
```

```
'A_A'
```

property whitespace

Whitespace rules.

Type
WhiteSpaceRules

class graphtransliterator.**CoverageTransliterator**(**args*, ***kwargs*)

Subclass of GraphTransliterator that logs visits to graph and on_match rules.

Used to confirm that tests cover the entire graph and onmatch_rules.

check_coverage(*raise_exception=True*)

Check coverage of graph and onmatch rules.

First checks graph coverage, then checks onmatch rules.

check_onmatchrules_coverage(*raise_exception=True*)

Check coverage of onmatch rules.

clear_visited()

Clear visited flags from graph and onmatch_rules.

Bundled Transliterators

graphtransliterator.transliterators

Bundled transliterators are loaded by explicitly importing *graphtransliterator.transliterators*. Each is an instance of graphtransliterator.bundled.Bundled.

class graphtransliterator.transliterators.**Bundled**(**args*, ***kwargs*)

Subclass of GraphTransliterator used for bundled Graph Transliterator.

property directory

Directory of bundled transliterator, used to load settings.

from_JSON(*check_ambiguity=False*, *coverage=False*, ***kwargs*)

Initialize from bundled JSON file (best for speed).

Parameters

- **check_ambiguity** (*bool*,) – Should ambiguity be checked. Default is *False.*
- **coverage** (*bool*) – Should test coverage be checked. Default is *False*.

from_YAML(*check_ambiguity=True*, *coverage=True*, ***kwargs*)

Initialize from bundled YAML file (best for development).

Parameters

- **check_ambiguity** (*bool*,) – Should ambiguity be checked. Default is *True.*
- **coverage** (*bool*) – Should test coverage be checked. Default is *True*.

generate_yaml_tests(*file=None*)

Generates YAML tests with complete coverage.

Uses the first token in a class as a sample. Assumes for onmatch rules that the first sample token in a class has a unique production, which may not be the case. These should be checked and edited.

load_yaml_tests()

Iterator for YAML tests.

Assumes tests are found in subdirectory *tests* of module with name *NAME_tests.yaml, e.g. `source_to_target/tests/source_to_target_tests.yaml*.

property name

Name of bundled transliterator, e.g. 'Example'

classmethod new(*method='json'*, ***kwargs*)

Return a new class instance from method (json/yaml).

Parameters

method (*str* (*json* or *yaml*)) – How to load bundled transliterator, JSON or YAML.

run_tests(*transliteration_tests*)

Run transliteration tests.

Parameters

transliteration_tests (*dict* of {*str*:*str*}) – Dictionary of test from source -> correct target.

run_yaml_tests()

Run YAML tests in MODULE/tests/MODULE_tests.yaml

property yaml_tests_filen

Metadata of transliterator

Type

dict

class graphtransliterator.transliterators.**Example**(***kwargs*)

Example Bundled Graph Transliterator.

class graphtransliterator.transliterators.**ITRANSDevanagariToUnicode**(***kwargs*)

ITRANS Devanagari to Unicode Transliterator.

class graphtransliterator.transliterators.**MetadataSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for Bundled metadata.

graphtransliterator.transliterators.**iter_names**()

Iterate through bundled transliterator names.

graphtransliterator.transliterators.**iter_transliterators**(***kwds*)

Iterate through instances of bundled transliterators.

Graph Classes

class graphtransliterator.**DirectedGraph**(*node=None*, *edge=None*, *edge_list=None*)

A very basic dictionary- and list-based directed graph. Nodes are a list of dictionaries of node data. Edges are nested dictionaries keyed from the head -> tail -> edge properties. An edge list is maintained. Can be exported as a dictionary.

node

List of node data

Type
list of *dict*

edge

Mapping from head to tail of edge, holding edge data

Type
dict of {*int*: *dict* of {*int*: *dict*}}

edge_list

List of head and tail of each edge

Type
list of *tuple* of (*int*, *int*)

Examples

```
from graphtransliterator import DirectedGraph
DirectedGraph()
```

```
<graphtransliterator.graphs.DirectedGraph at 0x7f250c24c500>
```

add_edge (*head*, *tail*, *edge_data=None*)

Add an edge to a graph and return its attributes as dict.

Parameters

- **head** (*int*) – Index of head of edge
- **tail** (*int*) – Index of tail of edge
- **edge_data** (*dict*, default {}) – Edge data

Returns
Data of created edge

Return type
dict

Raises
ValueError – Invalid `head` or `tail`, or `edge_data` is not a dict.

Examples

```
g = DirectedGraph()
g.add_node()
```

```
(0, {})
```

```
g.add_node()
```

```
(1, {})
```

```
g.add_edge(0,1, {'data_key_1': 'some edge data here'})
```

```
{'data_key_1': 'some edge data here'}
```

```
g.edge
```

```
{0: {1: {'data_key_1': 'some edge data here'}}}
```

add_node (*node_data=None*)

Create node and return (*int*, *dict*) of node key and object.

Parameters
node_data (*dict*, default {}) – Data to be stored in created node

Returns
Index of created node and its data

Return type
tuple of (*int*, *dict*)

Raises
ValueError – `node_data` is not a `dict`

Examples

```
g = DirectedGraph()
g.add_node()
```

```
(0, {})
```

```
g.add_node({'datakey1': 'data value'})
```

```
(1, {'datakey1': 'data value'})
```

```
g.node
```

```
[{}, {'datakey1': 'data value'}]
```

class graphtransliterator.**VisitLoggingDirectedGraph**(*graph*)

A DirectedGraph that logs visits to all nodes and edges.

Used to measure the coverage of tests for bundled transliterators.

check_coverage(*raise_exception=True*)

Checks that all nodes and edges are visited.

Parameters

raise_exception (*bool*, default) – Raise IncompleteGraphCoverageException (default, *True*)

Raises

IncompleteGraphCoverageException – Not all nodes/edges of a graph have been visited.

clear_visited()

Clear all visited attributes on nodes and edges.

Rule Classes

class graphtransliterator.**TransliterationRule**(*production*, *prev_classes*, *prev_tokens*, *tokens*, *next_tokens*, *next_classes*, *cost*)

A transliteration rule containing the specific match conditions and string output to be produced, as well as the rule's cost.

production

Output produced on match of rule

Type

str

prev_classes

List of previous token classes to be matched before tokens or, if they exist, *prev_tokens*

Type

list of *str*, or *None*

prev_tokens
List of tokens to be matched before *tokens*

Type
list of *str*, or *None*

tokens
List of tokens to match

Type
list of *str*

next_tokens
List of tokens to match after *tokens*

Type
list of *str*, or *None*

next_classes
List of tokens to match after *tokens* or, if they exist, *next_tokens*

Type
list of *str*, or *None*

cost
Cost of the rule, where less specific rules are more costly

Type
float

class graphtransliterator.**OnMatchRule**(*prev_classes*, *next_classes*, *production*)
Rules about adding text between certain combinations of matched rules.

When a translation rule has been found and before its production is added to the output, the productions string of an OnMatch rule is added if previously matched tokens and current tokens are of the specified classes.

prev_classes
List of previously matched token classes required

Type
list of *str*

next_classes
List of current and following token classes required

Type
list of *str*

production
String to added before current rule

Type
str

class graphtransliterator.**WhitespaceRules**(*default*, *token_class*, *consolidate*)
Whitespace rules of GraphTransliterator.

default
Default whitespace token

Type
str

token_class

Whitespace token class

Type
str

consolidate

Consolidate consecutive whitespace tokens and render as a single instance of the specified default whitespace token.

Type
bool

Exceptions

exception graphtransliterator.**GraphTransliteratorException**

Base exception class. All Graph Transliterator-specific exceptions should subclass this class.

exception graphtransliterator.**AmbiguousTransliterationRulesException**

Raised when multiple transliteration rules can match the same pattern. Details of ambiguities are given in a `logging.warning()`.

exception graphtransliterator.**NoMatchingTransliterationRuleException**

Raised when no transliteration rule can be matched at a particular location in the input string's tokens. Details of the location are given in a `logging.warning()`.

exception graphtransliterator.**UnrecognizableInputTokenException**

Raised when a character in the input string does not correspond to any tokens in the GraphTransliterator's token settings. Details of the location are given in a `logging.warning()`.

Schemas

class graphtransliterator.**DirectedGraphSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for *DirectedGraph*.

Validates graph somewhat rigorously.

class graphtransliterator.**EasyReadingSettingsSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for easy reading settings.

Provides initial validation based on easy reading format.

class graphtransliterator.**GraphTransliteratorSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for Graph Transliterator.

class graphtransliterator.**OnMatchRuleSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for *OnMatchRule*.

class graphtransliterator.**SettingsSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for settings in dictionary format.

Performs validation.

class graphtransliterator.**TransliterationRuleSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for *TransliterationRule*.

class graphtransliterator.**WhitespaceDictSettingsSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for Whitespace definition as a *dict*.

class graphtransliterator.**WhitespaceSettingsSchema**(*, *only: Sequence[str] | AbstractSet[str] | None = None*, *exclude: Sequence[str] | AbstractSet[str] = ()*, *many: bool = False*, *context: dict | None = None*, *load_only: Sequence[str] | AbstractSet[str] = ()*, *dump_only: Sequence[str] | AbstractSet[str] = ()*, *partial: bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown: str | None = None*)

Schema for Whitespace definition that loads as `WhitespaceRules`.

3.2.9 Credits

Development Lead

- A. Sean Pue @seanpue <pue@msu.edu>

Contributors

- Valentino Constantinou @vc1492a
- Rebecca Sutton Koeser @rlskoeser

3.2.10 Acknowledgements

Software development was supported by an Andrew W. Mellon Foundation New Directions Fellowship (Grant Number 11600613) and by matching funds provided by the College of Arts and Letters, Michigan State University.

3.2.11 Kudos

Graph Transliterator's developers acknowledge the following open-access projects, which have been particularly helpful in Graph Transliterator's development. These include: astropy (guide for documentation style expanding on numpy), click (command line interface), contributor_covenant (basis for the code of conduct), cookiecutter-pypackage (initial Python module template), jupyter-sphinx (renderer of live code results in the docs), and marshmallow (object serializer/deserializer).

Those from which code/text has been adopted are mentioned in NOTICE.

[~ Dependencies scanned by PyUp.io ~]

3.2.12 History

[Unreleased - Maybe]

- save match location in tokenize using token_details
- allow insertion of transliteration error messages into output
- fix Devanagari output in Sphinx-generated Latex PDF
- add translated messages
- add static typing with mypy
- adjust IncorrectVersionException to only consider major, minor versioning not patch
- Adjust CSS for CLI output in docs
- add doc making test to commit

[To do]

- Add on/off switch characters
- Update module publication

1.2.4 (2023-10-15)

- switched to poetry for module publishing, based on cookiecutter-poetry
- fixed click version command
- added code coverage

1.2.3 (2023-10-09)

- added python 3.10, 3.11, removed <=3.8
- updated dependencies (used pur)
- updated jupyter-download syntax
- reformatted with black
- adjusted flake8 line length
- removed collect_ignore for pytest
- updated Github actions

1.2.2 (2021-08-11)

- updated CONTRIBUTING.rst for new Python versions
- added github actions to publish to pypi and testpypi
- shifted to github CI
- updated dependencies
- fixed tox.ini
- updated schema.py error message
- updated docs/conf.py for jupyter_sphinx

1.2.1 (2020-10-29)

- updated docs/conf.py for jupyter_sphinx

1.2.0 (2020-05-13)

- changes to bundled.py and cli.py with dump-tests command
- updated cli.rst

1.1.2 (2020-04-29)

- updated LICENSE, minor code updates, security updates

1.1.1 (2020-04-21)

- Added test to check compressed dump is uniform
- Fixed sorting of class id in compressed dump to make JSON output uniform
- Added Python 3.8 support

1.1.0 (2020-01-10)

- Added pre-commit hook to rebuild bundled transliterators with bump2version
- remove to_dict from DirectedGraph, since it is handled through Marshmallow schemas.
- Adjust documentation to mention compression.
- added list-bundled CLI command
- added –regex/-re flag to graphtransliterator make-json CLI command to allow regular expressions
- removed coverage keyword from GraphTransliterator
- reorganized core.py
- converted from_dict, from_easyreading_dict, from_yaml, and from_yaml_file to static methods from class methods
- moved ambiguity-checking functions to ambiguity.py and tests to test_ambiguity.py

- set three levels of compression: 0 (Human-readable), 1 (no data loss, includes graph), 2 (no data loss, and no graph); 2 is fastest and set to default.
- set check_ambiguity to read keyword during JSON load
- allowed empty string productions during JSON compression
- added compression.py with decompress_config() and compress_config() to compress JSON
- added tests/test_compression.py to test compression.py
- added sorting of edge_list to DirectedGraph to allow dumped JSON comparison in tests
- adjusted _tokenizer_string_from() to sort by length then string for JSON comparison

1.0.7 (2019-12-22)

- added IncorrectVersionException, if serialized version being loaded is from a later version than the current graph-transliterator version
- added automatic edge_list creation if edge parameter in DirectedGraph
- added fields to and started using NodeDataSchema
- added pre_dump to GraphTransliteratorSchema, NodeDataSchema to remove empty values to compress Serialization
- removed rule from graph leaves and updated docs accordingly

1.0.6 (2019-12-15)

- fixed serialization of graph node indexes as integer rather than strings

1.0.5 (2019-12-14)

- added JOSS citation to README
- added –version to cli
- removed some asserts
- removed rule dictionaries from graph leaves to compress and simplify serialization

1.0.4 (2019-11-30)

- updates to docs

1.0.3 (2019-11-30)

- update to paper

1.0.2 (2019-11-30)

- updates for Zenodo

1.0.1 (2019-11-29)

- updated requirements_dev.txt

1.0.0 (2019-11-26)

- removed extraneous files
- updated development status in setup.py
- set to current jupyter-sphinx

0.4.10 (2019-11-04)

- fixed typo in requirements_dev.txt

0.4.9 (2019-11-04)

- quick fix to requirements_dev.txt due to readthedocs problem with not reading changes

0.4.8 (2019-11-04)

- twine update to 2.0

0.4.7 (2019-11-04)

- temp switch back to dev version of jupyter-sphinx for overflow error
- Dropped Python 3.5 support for twine 2.0 update

0.4.6 (2019-11-04)

- switched to latest jupyter-sphinx
- travis adjustments

0.4.5 (2019-10-31)

- Adjusted make-json CLI test to restore original example.json

0.4.4 (2019-10-24)

- moved README.rst to include in index.rst
- fixed error in advanced_tutorial.rst

0.4.3 (2019-10-24)

- fixed requirements_dev.txt

0.4.2 (2019-10-24)

- fixed README.rst for PyPI

0.4.1 (2019-10-24)

- fixed links to code in docs
- fixed link to NOTICE
- added acknowledgements

0.4.0 (2019-10-24)

- added bundled transliterators to api.rst
- adjustments to usage.rst
- adjustments to tutorial.rst
- fixes to docs (linking module)
- adjustments to advanced_tutorial.rst
- adjustments to README.rst
- fixes to AUTHORS.rst
- added kudos.rst to docs to acknowledge inspirational projects
- added advanced tutorial on bundling a transliterator.
- added cli.rst to docs
- fixed regex in get_unicode_char to allow hyphen
- added cli.py and adjusted setup.py
- updated tutorial
- added statement of need to README. Thanks @rlskoeser.
- Removed continue-linenos jupyter-sphinx directive in favor of configuration settings
- added preface to documentation source files with links to production version, etc. Thanks @rlskoeser.

- added custom css for jupyter-sphinx cells
- added jupyter-sphinx documentation with line numbering
- removed pkg_resources as source for version due to problem with loading from pythonpath for jupyter-sphinx in readthedocs, instead used __version__
- adjust path in docs/conf.py to fix docs error
- added bundled/schemas.py with MetadataSchema for bundled transliterator metadata
- added coverage to from_dict()
- added allow_none in onmatch_rules in GraphTransliteratorSchema
- adjusted core.py so that all edges are visited during search, even if no constraints
- removed _count_of_tokens() in favor of cost
- added IncompleteGraphCoverageException to exceptions.py
- added VisitLoggingDirectedGraph to graphs.py
- added tests/test_transliterator.py
- partially updated transliterators/README.rst
- removed transliterators/sample/*
- added yaml and json to package_data in setup.py
- Added to core.py class CoverageTransliterator, which tracks visits to edges, nodes, and onmatch rules, and allows clearing of visits and checking of coverage, used to make sure tests are comprehensive
- created test/test_coverage.py to test CoverageTransliterator
- created transliterators/bundled.py with class Bundled for bundled transliterators
- added load_from_YAML() and load_from_JSON() initializers to Bundled to load from bundled YAML (for development) and JSON (for speed)
- added load_yaml_tests(), run_yaml_tests(), and run_tests() to Bundled
- created transliterators/__init__.py that finds bundled transliterators in subdirectory and adds them to graphtransliterators.transliterators namespace
- added iter_names() and iter_transliterators() to transliterators/__init__.py
- created test/test_transliterator.py to check bundled transliterator loading and functions
- created in transliterators/example/ __init__.py, example.json, example.yaml
- created in transliterators/example/tests test_example.py and example_tests.yaml

0.3.8 (2019-09-18)

- fixed load() docstring example
- updated check_ambiguity() to use cost

0.3.7 (2019-09-17)

- Adjusted docs to show readme as first page
- Added sample graph and code to README.rst
- moved images in docs to _static

0.3.6 (2019-09-17)

- adjusted installation.rst renaming libraries to modules
- updated paper and bibliography.

0.3.5 (2019-09-15)

- flake8 fix for core.py
- fixed bug in schemas.py whereby, during load(), DirectedGraphSchema() was modifying input settings
- added tests for modifications to settings by load()
- adjusted DirectedGraphSchema to allow for compacted transliteration rule settings
- adjusted GraphTransliteratorSchema to allow for compacted settings
- added tests to confirm all optional fields passed to load() are really optional
- added ValidationError if onmatch_rules_lookup present without onmatch_rules
- adjusted DirectedGraphSchema edge definition to remove str if loading from JSON
- added more rigorous schema definitions for edge_list and node in DirectedGraphSchema
- fixed flake8 warning in graphs.py
- adjusted docstrings in core.py for dump(), dumps(), load(), and loads()

0.3.4 (2019-09-15)

- added sphinx-issues and settings to requirements_dev.txt, docs/conf.py
- added .readthedocs.yml configuration file to accommodate sphinx-issues
- removed history from setup.py due to sphinx-issues
- fixed GraphTransliteratorException import in __init__.py
- added docs/_static directory
- fixed emphasis error and duplicate object description in docs/usages.rst
- fixed docstring in core.py
- added python versions badge to README.rst (openjournals/joss-reviews#1717). Thanks @vc1492a.
- added NOTICE listing licenses of open-source text and code
- added Dependencies information to docs/install.rst (openjournals/joss-reviews#1717). Thanks @vc1492a.
- updated AUTHORS.rst
- minor updates to README.rst

0.3.3 (2019-09-14)

- fixed missing marshmallow dependency (#47). Thanks @vc1492a.
- removed unused code from test (#47). Thanks @vc1492a.
- removed cerberus dependency

0.3.2 (2019-08-30)

- fixed error in README.rst

0.3.1 (2019-08-29)

- adjustments to README.rst
- cleanup in initialize.py and core.py
- fix to docs/api.rst
- adjusted setup.cfg for bumpversion of core.py
- adjusted requirements.txt
- removed note about namedtuple in dump docs
- adjusted docs (api.rst, etc.)

0.3.0 (2019-08-23)

- Removed _tokens_of() from init
- Removed serialize()
- Added load() to GraphTransliterator, without ambiguity checking
- Added dump() and dumps() to GraphTransliterator to export configuration
- renamed _tokenizer_from() to _tokenizer_pattern_from(), and so that regex is compiled on load and passed as pattern string (tokenizer_pattern)
- added settings parameters to DirectedGraph
- added OnMatchRule as namedtuple for consistency
- added new GraphTransliterator.from_dict(), which validates from_yaml()
- renamed GraphTransliterator.from_dict() to GraphTransliterator.from_easyreading_dict()
- added schemas.py
- removed validate.py
- removed cerberus and added marshmallow to validate.py
- adjusted tests
- Removed check_settings parameter

0.2.14 (2019-08-15)

- minor code cleanup
- removed yaml from validate.py

0.2.13 (2019-08-03)

- changed setup.cfg for double quotes in bumpversion due to Black formatting of setup.py
- added version test

0.2.12 (2019-08-03)

- fixed version error in setup.py

0.2.11 (2019-08-03)

- travis issue

0.2.10 (2019-08-03)

- fixed test for version not working on travis

0.2.9 (2019-08-03)

- Used Black code formatter
- Adjusted tox.ini, contributing.rst
- Set development status to Beta in setup.py
- Added black badge to README.rst
- Fixed comments and minor changes in initialize.py

0.2.8 (2019-07-30)

- Fixed ambiguity check if no rules present
- Updates to README.rst

0.2.7 (2019-07-28)

- Modified docs/conf.py
- Modified equation in docs/usage.rst and paper/paper.md to fix doc build

0.2.6 (2019-07-28)

- Fixes to README.rst, usage.rst, paper.md, and tutorial.rst
- Modifications to core.py documentation

0.2.5 (2019-07-24)

- Fixes to HISTORY.rst and README.rst
- 100% test coverage.
- Added draft of paper.
- Added graphtransliterator_version to serialize().

0.2.4 (2019-07-23)

- minor changes to readme

0.2.3 (2019-07-23)

- added xenial to travis.yml

0.2.2 (2019-07-23)

- added CI

0.2.1 (2019-07-23)

- fixed HISTORY.rst for PyPI

0.2.0 (2019-07-23)

- Fixed module naming in docs using __module__.
- Converted DirectedGraph nodes to a list.
- Added Code of Conduct.
- Added GraphTransliterator class.
- Updated module dependencies.
- Added requirements.txt
- Added check_settings parameter to skip validating settings.
- Added tests for ambiguity and *check_ambiguity* parameter.
- Changed name to Graph Transliterator in docs.
- Created core.py, validate.py, process.py, rules.py, initialize.py, exceptions.py, graphs.py
- Added ignore_errors property and setter for transliteration exceptions (UnrecognizableInputToken, NoMatchingTransliterationRule)

- Added logging to graphtransliterator
- Added positive cost function based on number of matched tokens in rule
- added metadata field
- added documentation

0.1.1 (2019-05-30)

- Adjusted copyright in docs.
- Removed Python 2 support.

0.1.0 (2019-05-30)

- First release on PyPI.

3.3 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

V

W

Y