Minority Multilingual Acoustic-Character Processing Integrated Framework

Natural language processing and corpora compilation works for less popular minority languages are time consuming and difficult labor. Standardization, maintenance, and research capital are scarce for these less popular languages, which cause more difficulties for collecting high quality corpora. Furthermore, the repeated works on corpora preparation, less co-operation, and incompatibility problems urge us to develop a relatively comprehensive and compact framework to improve public participation into this work, together we hope to make it publicly available resource.

This tool is designed to reduce repeated code-lines as much as possible. There are three layers in this framework: (1) The parent layer is the multilingual phoneme-character processing class: Mchar() which works on character normalization and acoustic rule analysis. (2) Middle layers are language specific classes which finish language specific works like irregular phonetics, specific spelling rules, syllable templates etc. we should keep this layer as small as possible. (3) This layer provides user&file interface class: Mcount(). Files and strings are processed from this layer to transform and normalize character codes, to produce acoustic dictionary, pronunciation files, and other particles like words, syllables, and characters. It can also be used for rule-based spell-checking.

(1) Layer one [Class Mchar()] this is a language independent layer. Please utilize this layer as much as possible, try not to overlay methods of this class in order to keep the language-specific middle layer small. Generally this layer has 4 works to do: (a) Code_init() method initialize character-codes by loading code-map-file for a certain language and provide a uniform ASCII basic codes for all layers. The main purpose is to normalize various codes (there are multi-code-forms for every character in these languages) into a unified code for a specific language. (b) Code_flip() method will transform a source-text-file of various codes into the target unified ASCII codes or vice versa. (c) Syll_split() method segment strings into syllables for almost any languages if the syllable template is given by layer2. This method can produce all possible segmentation results, and the first candidate is more conventional. It is also used to spell-checking according to syllable structures. Please use this method instead of creating a language specific tool. (d) Some methods for checking alphabets and vowels of a string for various languages which need only to provide alphabet code-ranges and a vowel set.

(2) Layer two consists of various language specific classes [Class: xxLANGchar()] which inherit first layer [Mchar()]. Each language has its own class, please try to make this layer as small as possible by utilizing other layers. Some languages like Uyghur has simpler spelling rules, and clear correspondence between phonetics and characters, so no need to have much language specific programs. But some languages like Kazak have a set of more complicated rules to implement.

(3) Layer three [class Mcount(lang)] is an independent class, and provides user and file interfaces. There is an object “charObj” inside to utilize other layers according to certain language name “lang”. Users can process their multilingual text files or strings from this language independent layer without knowing about any specific language. But, there are some user defined particle-labels like “_”or “-” which can be used to label morphemes or as a label of word boundary . This layer provides some functions: (a) Code_transform() method works on text files, and transforms certain character-codes of certain language into ASCII codes via “charObj”. (b) Line_process() is a generative method that will process each line of a file to generate various particle of words, syllables, characters, acronyms, unknown-characters etc. Other method use this to process each line of text. (c) Token_Vocab()

method collect particles with their frequencies from a file and save them to a set of dictionary variables. Then the Particle_export() method will export these particles as a list of vocabulary. Or users can change the export format to their convenience. (d) File_export() method exports acoustic files, acoustic dictionary, or other particle sequence-files.

We also provide some recipe and test-text-files as a simple instructor. We hope anyone could participate, improve, and expend this tool to include more languages and functions.

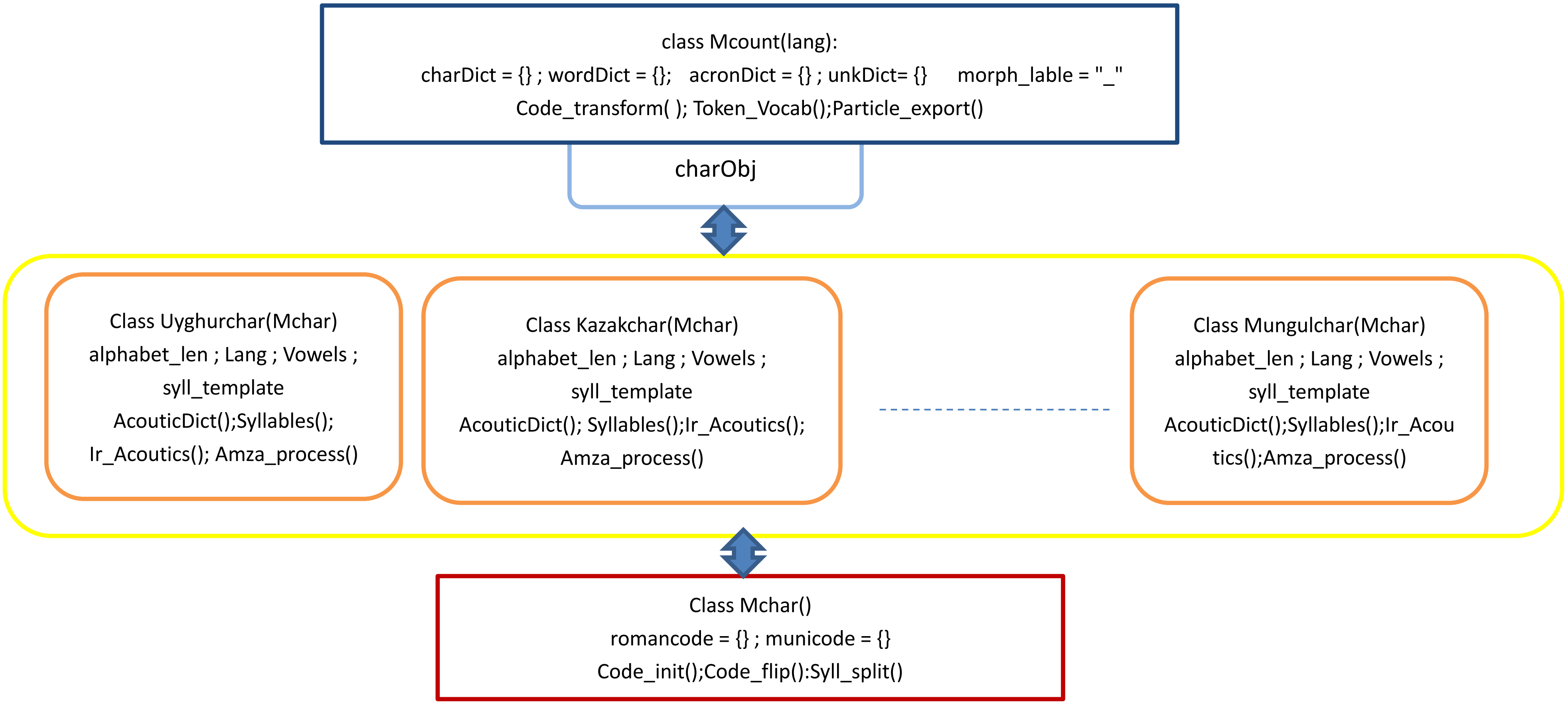
class Mcount(lang):
charDict = {} ; wordDict = {}; acronDict = {} ; unkDict= {} morph_lable = "_"
Code_transform( ); Token_Vocab();Particle_export()
charObj
Class Uyghurchar(Mchar)
alphabet_len ; Lang ; Vowels ;
syll_template
AcouticDict();Syllables();
Ir_Acoutics(); Amza_process()
Class Kazakchar(Mchar)
alphabet_len ; Lang ; Vowels ;
syll_template
AcouticDict(); Syllables();Ir_Acoutics();
Amza_process()
Class Mungulchar(Mchar)
alphabet_len ; Lang ; Vowels ;
syll_template
AcouticDict();Syllables();Ir_Acou
tics();Amza_process()
Class Mchar()
romancode = {} ; municode = {}
Code_init();Code_flip():Syll_split()

Image 1 Minority Multilingual Acoustic-Character Processing Integrated Framework

"m2char" folder contains a character and acoustic processing framework for multi-languages.
the contents are:
./doc: manual files which describe this tool.
./rc: code-map-files for various languages, used for character transformation from unicodes to ASCII codes.
./ut: a unit test folder.

This tool accomplish code transformation for various less popular languages whose character-codes are diverse,
and necessary to be unified into un-ambiguous codes. Or , we can call it character normalization.

As a part of ASR project this tool can produce acoustic dictionary and acoustic files for various languages.

This tool also include spell-checking for various languages according to fundamental rules like: syllable structure, pronunciation rules, ect.