MICROMEDIA INTERNATIONAL telecommunication & multimedia	***Micromedia International***	**Specification**

Author : Pierre Chevrier **Pages : 12**

Company : Micromedia International

Date : 25/08/2010

Vocalizer dictionary and rules

Ref. : ETT_20100825_000001.docx

This document describes the user dictionary and the user rules file format.It is a part of the Vocalizer User Guide.

Diffusion	A	CC	Company

Attached documents :

Contents

User Dictionaries

Introduction

User dictionaries allow you to specify special pronunciations for particular words or multi-word strings. They make it possible to customize the output of the Text-To-Speech system. In particular, a user dictionary contains mappings from an orthographic string to either a phonetic transcription, or to an orthographic transcription, e.g. to expand abbreviations.

Phonetic transcriptions in a particular language are composed of phonemes represented by L&H+ symbols. More information about these phonemes can be found in the **User's Guide for <Language>.**

You create a user dictionary in a text file with extension (*.tdc) and you compile it as a binary format (*.bdc). Then you load the binary user dictionary on Vocalizer.

The tool used to compile the dictionary is dictcpl.exe. Run it in a command shell.

Usage : *dictcpl.exe [options] 'tdc-file'*
Options : -o bdc-file

Vocalizer consults the user dictionary for each individual word in the input text, and for multi-word fragments tagged by the control sequence <ESC>\mw\. First it looks up the string "as is", then the string with leading and trailing quotes and brackets stripped, then with trailing dots stripped, then the string in lower case. It tries these candidates until the lookup returns a hit, or all are missed.

Text format description

A text user dictionary is a plain Unicode text file (encoded in UTF-16 or UTF-8). It contains one or more data sections, each section being a collection of entries that share the same attributes. A section is defined in particular by:

- a header section specifying attributes such as language and replacement type, and
- a number of user dictionary entries.

For instance

[Subheader]
Language = <language_code>
Content = <content_spec>
Representat ion = <repr_spec>
[Data]
<entries>

A user dictionary entry is basically a key-value pair. The key is the word or multi-word string, and the value is its replacement string, i.e. an orthographic or a phonetic transcription, e.g.

DLL "Dynamic Link Library"

A text user dictionary has to start with a file header section marked with the string "[Header]". The following fields go in that header:

- Language: mandatory field: the language of the entries.
- Name: optional field: a name for the dictionary.
- Description: optional field: a description of what is in the dictionary.
- Content: optional field: the type of entries stored in the following section (phonetic transcriptions or orthographic transcriptions).
- Representation: optional field: the representation of the entries in the following sections.

For instance

[Header]
Language = ENG
Content = EDCT_CONTENT_BROAD_NARROWS
Representation = EDCT_REPR_SZZ_STRING

After the file header section, one or more data sections can be specified. Each data section has a "[SubHeader]" and a "[Data]" part, as illustrated above. If the user dictionary contains only one data section, the "[SubHeader]" section can be omitted (and the entries take the attributes from the file header).

The "[SubHeader]" part can have the same fields as the file header marked by "[Header]". For each attribute, the rule applies that a value specified at some point overrules the values specified earlier. So if a "[SubHeader]" part provides a value for an attribute, this becomes the new value for the rest of the user dictionary until it is assigned a new value or until the end of the file. This also means that the value specified in the "[Header]" section is an initial value; it is not the default value for all data sections.

There are two exceptions to this rule: the "Name" and "Description" fields apply to the entire dictionary, and they take their value from the last specified value.

Although the format supports several subheaders with different values for the fields, a user dictionary only supports a single value for the "Language" field.

The "Name" and "Description" are free format text fields to be defined by the user. You can use them for your own convenience.

The "Language" value is the three letter language code used by Vocalizer, e.g.: ENU for "American English". See the Language Codes table above for a list.

The values for the "Content" and "Representation" go together:

- Phonetic transcription are marked by EDCT_CONTENT_BROAD_NARROWS and EDCT_REPR_SZZ_STRING.
- Orthographic replacements are marked by EDCT_CONTENT_ORTHOGRAPHIC and EDCT_REPR_SZ_STRING.

The "[Data]" part contains one entry per line:

- The key and value are separated by white space. Double quotes can be used around key and value in case they contain spaces. In that case, the backslash can be used as an escape character for a literal double quote and a literal backslash.
- The dictionary keys are case sensitive.
- An orthographic replacement value is a regular text string, a phonetic replacement starts with the tag "//" followed by the transcription in L&H+ format.

It's important to put one or more space characters between the key and the value, and not to put unintentionally spaces after the value (as this will be considered part of the value).

Text format specification

dictionary:
header data |
header (subheader data)+

header:
[Header]
attribute+

subheader:
[SubHeader]
attribute+

attribute:
Name = <string> |
Language= <3 letter code> |
Description = <string> |
Content = [
EDCT_CONTENT_BROAD_NARROWS |
EDCT_CONTENT_ORTHOGRAPHIC
] |
Representation = [
EDCT_REPR_SZZ_STRING |
EDCT_REPR_SZ_STRING
]

data:
[Data]
*(key value)**

key:
<string> |
<quoted string>

value:
phonetic |
orthographic

phonetic:
//<string>

orthographic:

<string> |
<quoted string>

Example text user dictionary

This is how a text user dictionary might look like:

[Header]
Language = ENG

[SubHeader]
Content = EDCT_CONTENT_BROAD_NARROWS
Representation = EDCT_REPR_SZZ_STRING

[Data]
zero // #'zi .R+o&U#
addr // #'@.dR+Es#
adm // #@d. '2mI .n$. 'stR+e&I .S$n#

[SubHeader]
Content=EDCT_CONTENT_ORTHOGRAPHIC
Representation=EDCT_REPR_SZ_STRING

[Data]

Info	*Information*
IT	*"Information Technology"*
DLL	*"Dynamic Link Library"*
A-level	*"advanced level"*
Afr	*Africa*
Acc	*account*
TEL	*telephone*
Anon	*anonymous*
AP	*"associated press"*

User Rulesets

Introduction

Rulesets allow the user to specify "search-and-replace" rules for certain strings in the input text. Whereas user dictionaries only support search and replace functionality for literal strings that are complete words or tagged multi-word fragments, rulesets support any search pattern that can be expressed using regular expressions (e.g. multiple words, part of a word).

The rulesets are applied before any other text normalization is performed, including user dictionary lookup.

The details of how the text normalization can be tuned via user rulesets are described in the next section.

A ruleset is basically a collection of rules; each rule specifies a “search pattern” and the corresponding “replacement spec”.

The syntax and semantics of the “search pattern” and the “replacement spec” match those of the regular expression library that is used, being PCRE v5.0 which corresponds with the syntax and semantics of Perl 5. For the Perl 5 regular expression syntax, please refer to the Perl regular expressions main page at http://perldoc.perl.org/perlre.html. For a description of PCRE, a free regular expression library, see http://www.pcre.org/.

More details on the syntax are described in the “Ruleset format” section.

The rules of a loaded ruleset are applied only when the active language matches the language that is specified in the header section of the ruleset. Moreover, a user ruleset can be global in scope, or can be restricted to a block of text marked with a particular tn value (with the <ESC>\tn\ control sequence)

Ruleset format

Ruleset are described in a UTF-8 encoded text file.

In general, a ruleset consists of a header section, followed by a data section. The format of a ruleset is described formally below using the following notation:

Symbol	Meaning
{…}	Optional part; the part between { and } can be occur once but is not required to.
(…)*	The part between (and) can be occur more than once.
<…>	The part between < and > specifies a variable string constant.
A|B	OR part, A is specified or B is specified.

A ruleset can be formally described as:

```
ruleset :=
        (<comment-line>|<blank-line>)*
        <header-section>
        <data-section>?
```

Comment lines have the '#' character as the first non-blank character.

A blank line is a line consisting entirely of linear whitespace characters. Using regular expression syntax they can be expressed as:

comment-line := ^\s*#.*\n

blank-line := ^\s*\n

Header Section

The "header" section contains one or more key definitions (the definition of the "language" key is required, see further); each definition can span one line only.

header-section :=
"[header]"\n
(<comment-line>|<blank-line>|
<key-definition>)+

Comment lines and blank lines can be inserted everywhere.

Key definitions have the following syntax:

key-definition :=<key-name> = <key-value><comment>?\n

Blanks (spaces or tabs) before and after the equal sign are optional.

If the key value contains blanks, it must be enclosed in double quotes. If a double quote is needed as part of the value, it needs to be escaped (\"). The actual syntax of the <key-value> depends on the <key-name>.

A <comment> can follow the <key-value>, it lasts until the end of the line.

comment :=#.*$

The only currently supported key names are: “language”, “charset” and “type”. This means that <key-definition> can be expressed semantically as:

key-definition :=<language-definition>|<charset-definition> | <type-definition>

The <language-definition> is required for each header, the value is the 3-letter Vocalizer language code, a language group or the wildcard ‘*’ for specifying all languages. The 3-letter language code is also used to specify the language of user dictionaries, see the Language Codes table above for a list.

Note that the “*” used in the following syntax specification designates the literal asterisk character “*”, and not a repetition.

language-definition :=
language =
(<language-code-list>|<language-group>|*)<comment>?\n

language-code-list := <language-code>(,<language-code>)*

language-code := ENA|ENG|ENU|DUN|FRC|GED|…

language-group := EN* | DU* | FR* | GE* | …

The <charset-definition> is optional and specifies the character set used for the encoding of the rules. Currently the character set must be UTF-8.

charset-definition :=charset = <charset id> <comment>? \n

charset id :="utf-8"

The type-definition is optional and specifies that the ruleset is scoped to text marked for a particular tn value (with the <ESC>\tn\ control sequence. A ruleset with a type-definition is global.

type-definition := type = <type-name><comment>?\n

The type-name is any non-white-space character sequence, and corresponds to the value of the <ESC>\tn\ control sequence. For example, a user ruleset with a type-name of “financial:stocks” can be accessed using <ESC>\tn=financial:stocks\.

Data Section

The "data" section contains zero or more "rules", a rule can occupy one line only.

data-section :=
    "[data]"\n
    (<comment-line>|<blank-line>|<rule>)*

Comments can also be inserted at the end of a rule and start with a '#'

character and span till the end of the line.

A rule has the following syntax:

rule := <search-spec> "-->" <replacement-spec> <comment>? \n

The syntax and semantics of the <search-spec> and the <replacement-spec> matches the one of the used regular expression library, being PCRE v5.0, this corresponds with the syntax and semantics of Perl 5. For Perl 5 regular expression syntax, please refer to the Perl regular expressions man page at http://perldoc.perl.org/perlre.html. For a description of PCRE, a free regular expression library, see http://www.pcre.org/.

For a detailed description, see the "pcrepattern.html" document in the PCRE distribution package.

If markup is being used (in the source and/or replacement pattern), it must be in the native Vocalizer markup format.

Note that special characters and characters with a special meaning need to be escaped.

Some examples are:

- In the search pattern: non-alphanumerical characters with a special meaning like dot(.), asterisk (*), dollar ($), backslash (\) and so on, need to be preceded with a backslash when used literally in a context where they can have a special meaning (e.g. use * for *). In the replacement spec this applies to characters like dollar ($), backslash (\) and double quote (").
- Control characters like \t (Tab), \n (Newline), \r (Return), etc.
- Character codes: \xhh (hh is the hexadecimal character code, e.g. \x1b for Escape), \ooo (ooo is the octal notation, e.g. \033 for Escape).
- Perl5 also predefines some patterns like “\s” (whitespace) and “\d” (numeric).

For a full description please refer to the Perl5 man pages.

Rule example

/David/ --> "Guru of the month May"

Replaces each occurrence of the string "David" by "Guru of the month May".

Search-spec

In general the format of the search-spec is:

Search-spec := <delimiter> <regular-expression> <delimiter> <modifier>*

<delimiter> is usually '/', but can be any non-whitespace character except for digits, back-slash ('\') and '#'... This facilitates the specification of a regular expression that contains '/', because it eliminates the need to escape the '/'.

<modifier> := [imsx]

Optional modifiers:

- i (search is case-insensitive);
- m (let '^' and '$' match even at embedded newline characters);
- s (let the '.' pattern match even at embedded newline characters, by default '.' matches any arbitrary character, except for a newline);
- x (allows for regular expression extensions like inserting whitespace and comments in <regular-expression>).

Replacement-spec

The format of the replacement spec is a quoted ("…") string or a non-blank string in case the translation is a single word. It may contain back references of the form $n (n: 1, 2, 3, ...) which refer to the actual match for the n-th capturing subpattern in <search-spec>. E.g. $1 denotes the first submatch. A back reference with a number exceeding the total number of submatches in <search-spec>, is translated into an empty string. A literal dollar sign ($) must be escaped (\$).

Everything following <replacement-spec> and on the same line is considered as comment when starting with '#', else it is just ignored.

Some rule examples

/<NUAN>/ --> "Nuance Communications"

Rewrites "<NUAN>" into "Nuance Communications".

/(Quack)/ --> ($1)

Replaces "Quack" by "(Quack)".

/(Quack)/ --> ($2)

Replaces "Quack" by "()".

/(\s):-\)(\s)/ --> "$1ha ha$2"

Where "\s" matches any whitespace character, $1 corresponds with the matched leading whitespace character and $2 corresponds with the matched trailing whitespace character. This rule rewrites for instance " :-) " into " ha ha ".

/(\r?\n)-{3,} *Begin included message *-{3,}(\r?\n)/ --> "$1Start of included message:$2"

Rewrites for instance ---- Begin included message ---- into Start of included message:

/\x80 ?(\d+)\.(\d{2})\d*/ --> "$1 euro $2 cents"

Rewrites for instance "€9.751" into "9 euro 75 cents".

Restrictions on rulesets

The following restriction applies to rulesets: markers generated while rulesets are loaded have source position fields that represent the position after the rulesets have been applied.

Effect of rulesets on performance

The loading of rulesets can effect synthesis processing performance, increasing latency (time to first audio) and overall CPU use. Certain regular expression patterns are more efficient than others, so it is important to carefully consider pattern efficiency while writing rulesets, and to test the system with and without the rulesets to ensure the performance is acceptable.

E.g. a character class (e.g. "[aeiou]") is more efficient than the equivalent set of alternatives (e.g. "(a|e|i|o|u)").

See the "pcreperform.html" main page of the PCRE package for more details.