Expressive Speech CORE

Text normaliser

Reference manual

Core version: 8.1

Reference	:	VOX341_Text_normaliser_manual_8.1_1.0_EN			
Doc. version	:	1.0	**Status**	:	release
Date	:	07/04/2017	**Diffusion**	:	restricted

Document review

Version	Date	Author	Verified by	Modification
1.0	07/04/2017	ER		Creation for Expressive Speech Core 8.1

Summary

1 Presentation

1.1 Overview

A normaliser rules file is used to preprocess the input text at the beginning of the synthesis pipeline of the text-to-speech engine. It may be used for any generic transformation you want to apply on the text to read.
The syntax of the normaliser rules file is described here. They may contain any number of rules which are applied one after another to the input text, the result of a rule being used as input for the next one.
In the case of an SSML document, the input text is split into tokens, whereby a token is a sequence of characters between SSML markups (except <mark> which doesn't split input text). Rules will be applied on individual tokens instead of being applied on whole text.
Whenever the *pattern* of a rule matches the input text, the matched subsequence is replaced with the *replacement* part of the rule.
Due to the streamed operation of the text-to-speech engine, there is a limit to the number of consecutive characters which may be replaced in a sequence :

- Rules are not guaranteed to match sequences over 256 characters: avoid using patterns like '.*' which may not match correctly in long text tokens.
- If cumulated rules application leads to a sequence of text over 768 characters of modified text or text without blanks or punctuation, some rules may not be correctly applied.

1.2 Terminology

Abbreviation	Description
SSML	*Speech Synthesis Markup Language*
TTS	*Text To Speech*
RGX rules/file	*Abbreviation for text normaliser rules/file*
Baratinoo	*Abbreviation for the Voxygen Expressive Speech Core (Voxygen TTS engine)*

1.3 Reference documents

Reference	Document name
V0X31	SSML reference manual

2 Normaliser syntax

2.1 Normaliser file format

A normaliser file must be a conformant XML document. Filename extension is .rgx+xml.

It is recommended to always specify the document's character encoding scheme with the `encoding` attribute in the `xml` prologue.

When an normaliser document is not compliant, the Baratinoo engine will issue notification of the problem encountered and may return an input error.

2.2 Normaliser elements

Here is a list of supported normaliser elements and their attributes.

The second column ("St") of the attribute tables indicates the status of each attribute:

`M`: mandatory (must/required; error if not present)
`O`: optional (may; ok if not present)

The normaliser markup consists of the following elements and attributes:

<normaliser>	version xmlns pattern_syntax pattern_options replacement_syntax	Root element for normaliser
<replace>		Container element for a rule
<pattern>	syntax options	Matching part of a rule
<replacement>	syntax	Replacement part of a rule

<normaliser>

Description

Root of an normaliser document.

Attribute	St	Value
`version`	`M`	`"1.0"`
`xmlns`	`M`	`"http://www.voxygen.fr/tts"`
`pattern_syntax`	`O`	Set pattern default syntax for all rules.
`pattern_options`	`O`	Set pattern default options for all rules.
`replacement_syntax`	`O`	Set replacement string default syntax for all rules.

Possible content
Zero or more `<replace>` elements.

<replace>

Description

Container for a rule entry. The <replace> element has no attributes.

Possible content
One <pattern> element and one <replacement> element.

<pattern>

Description

The <pattern> element contains text describing the part of the input the rule will match against.

Attribute	St	Value
syntax	0	Syntax of the regular expression “ECMAScript” (default) : modified ECMAScript regex grammar “basic”: basic POSIX regex grammar “extended”: extended POSIX regex grammar “awk”: regex grammar used by awk utility in POSIX “grep”: regex grammar used by grep utility in POSIX “egrep”: regex grammar used by grep utility with -E option
options	0	Options to apply to the rule, separated with spaces. “icase”: ignore case in character matching “nosubs”: all subexpressions are treated as non-marking subexpressions “collate”: ignore diacritics in character ranges ([a-z]) “optimize”: try to increase matching speed, may increase loading time.
say-as	0	String. Works with the SSML <say-as> element. If a say-as value is set, only text within a <say-as> element whose interpret-as attribute equals this value will be compared to this <pattern> element.

Possible content
Only text (must not be empty)

Note

The ECMAScript regular expressions syntax also accepts the following POSIX character class names: alpha, upper, lower, digit, xdigit, alnum, space, punct, print, graph and cntrl. That means you can use for example [[:digit:]] to match any digit in the input text.

The ECMAScript syntax also accepts character class equivalence. That means you can use for example [[=e=]] to match any form of 'e' character (e, é, ë…).

It is recommended to use the ECMAScript grammar.

Warning: Due to some variant in C++11 regex implementation, the symbols '^' and '$' do not always behave as expected. Do not use them if you need a consistent behavior over platforms and versions of the software.

<replacement>

Description

The <replacement> element contains text which will replace the part of the input that has been matched by the <pattern> element of the rule.

Attribute	St	Value
syntax	0	Syntax of the replacement string. "ECMAScript" (default). "sed".

Possible content
Only text (may be empty)

Note

You can use one of two different syntaxes to write replacement strings, no matter which syntax you used for <pattern> element:

- ECMAScript:

  The replacement text may refer to marked sub-expression in the <pattern> element by using $*n* or $*nn*, n being a number in the range [1-9] or [01-99] corresponding to the *n*th sub-expression in the <*pattern*> element.

  You can use $& to refer to the entire matched substring.

  You can use $$ to produce a single '$' character.

- Sed:

  The replacement text may refer to marked sub-expression in the <pattern> element by using *n* or *nn*, n being a number in the range [1-9] or [01-99] corresponding to the *n*th sub-expression in the <*pattern*> element.

  You can use & to refer to the entire matched substring.

  You can use \\ to produce a single '\' character.

  You can use \& to produce a '&' character.

  You can also include a special sequence made of a backslash and one of the letters L, l, U, u or E. The meaning is the same as in GNU sed:

  \L : turn the replacement to lowercase until a \U or \E is found

  \l : turn the next character to lowercase

  \U : turn the replacement to uppercase until a \L or \E is found

  \u : turn the next character to uppercase

  \E : Stop case conversion started by \L or \U

2.3 Example

Here is an example of a normaliser file:

```
<?xml version="1.0" encoding="utf-8"?>
<normaliser version="1.0" xmlns="http://www.voxygen.fr/tts"
pattern_syntax="ECMAScript" pattern_options="">
    <replace>
        <pattern>\$([[:digit:]]+)\.([[:digit:]]+)</pattern>
        <replacement>$1$$ and $2 cents</replacement>
    </replace>
    <replace>
        <pattern options="icase">([0-2]?[0-9])h([0-5]?
[[:digit:]])</pattern>
        <replacement>$1:$2</replacement>
    </replace>
    <replace>
        <pattern options="icase">([0-2]?[0-9])h</pattern>
        <replacement syntax=”sed”>\1:00</replacement>
    </replace>
    <replace><!-- removes all forms of 'e' -->
        <pattern options="collate">[[=e=]]</pattern>
        <replacement />
    </replace>
    <replace>
        <pattern>(\w+)</pattern><!-- matches a word -->
        <replacement syntax="sed">\L\u\1</replacement><!--
convert the word to lowercase except the first character which is
converted to uppercase -->
    </replace>
</normaliser>
```