Apertium stream format

En français

This page describes the stream format used in the Apertium machine translation platform.

Characters

Reserved

Reserved characters should only appear escaped in the input stream unless they are part of a lexical unit, chunk or superblank.

- The characters `^` and `$` are reserved for delimiting lexical units
- The character `/` is reserved for delimiting analyses in ambiguous lexical units
- The characters < and > are reserved for encapsulating tags
- The characters `{` and `}` are reserved for delimiting chunks
- The character `\` is the escape character

Special

The following have special meaning at the start of an analysis:

- Asterisk, `'*'` -- Unanalysed word.
- At sign, `'@'` -- Untranslated lemma.
- Hash sign, '#'
  - In morphological generation -- Unable to generate surface form from lexical unit (escape this to use # in lemmas)
  - In morphological analysis -- Start of invariable part of multiword marker (escape this to use # in lemmas)
- Plus symbol, '+' -- Joined lexical units (escape this to use + in lemmas)
- Tilde '~' -- Word needs treating by post-generator

Python parsing library

If you're writing a python script that needs to handle the Apertium stream format, try the excellent https://github.com/apertium/streamparser which lets you do

```
from streamparser import parse_file, mainpos, reading_to_string
for blank, lu in parse_file(file, with_text=True):
    analyses = lu.readings
    firstreading = analyses[0]
    surfaceform = lu.wordform
    # rewrite to print only the first reading (and surface/word form):
    print("^{}/{}$".format(surfaceform,
                           reading_to_string(firstreading)))
    # convenience function to grab the first part of speech of the first reading:
    mainpos = mainpos(lu)
```

etc. without having to worry about superblanks and escaped characters and such :-)

Here's an example used in testvoc, this one splits ambiguous readings like `^foo/bar<n>/fie<ij>$` into `^foo/bar<n>$ ^foo/fie<ij>$`, keeping the (super)blanks and newlines in between unchanged:

```
from streamparser import parse_file, reading_to_string
import sys
for blank, lu in parse_file(sys.stdin, with_text=True):
    print(blank+" ".join("^{}/{}$".format(lu.wordform, reading_to_string(r))
                         for r in lu.readings),
          end="")
```

Here's a one-liner to print the lemmas of each word:

```
$ echo fisk bank kake|lt-proc nno-nob.automorf.bin|python3 -c  'import sys, streamparser; print ("\n".join("\t".join(set(s.baseform for r in lu.readings for
s in r)) for lu in streamparser.parse_file(sys.stdin)))'
```

An alternative python lib: https://github.com/krvoje/apertium-transfer-dsl/blob/master/apertium/stream_entities.py https://github.com/krvoje/apertium-transfer-dsl/blob/master/apertium/stream_reader.py

Common Lisp parsing library

cl-apertium-stream[1] (https://github.com/veer66/cl-apertium-stream) is a library written in Common Lisp for parsing Apertium stream and generating Apertium stream from parsed data. It is developed based on the discontinued Ruby library[2] (https://github.com/veer66/reinarb). cl-apertium-stream is data-driven. Its parsed data is a list, keyword, and string combination without any new type/class. So further processing is based on ordinary list operations. cl-apertium-stream handles Apertium stream format by declarative Esrap[3] (https://github.com/scymtym/esrap) rules.

Formatted input

See also: Format handling

F = formatted text, T = text to be analysed.

Formatted text is treated as a single whitespace by all stages.

```
[<em>]this is[<\/em> ]a[ <b>]test.[][<\/b>]

|____|       |_______| |____|     |_______|
   |            |        |            |
   F            F        F            F

[<em>]this is[<\/em> ]a[ <b>]test.[][<\/b>]
      |______|        |      |____|
          |           |        |
          T           T        T
```

Analyses

S = surface form, L = lemma.

```
^vino/vino<n><m><sg>/venir<vblex><ifi><p3><sg>$

   |    | |________|
   S    L    TAGS
        |______|
        ANALISIS

|_____________________________________________|
          AMBIGUOUS LEXICAL UNIT

^vino<n><m><sg>$

|______________|
 DISAMBIGUATED
  LEXICAL UNIT

^dímelo/decir<vblex><imp><p2><sg>+me<prn><enc><p1><mf><sg>+lo<prn><enc><p3><nt>/decir<vblex><imp><p2><sg>+me<prn><enc><p1><mf><sg>+lo<prn><enc><p3><m><sg>$

                                 |____________________________________________|
                                               JOINED MORPHEMES
```

```
^take it away/take<vblex><sep><inf>+prpers<prn><obj><p3><nt><sg># away/take<vblex><sep><pres>+prpers<prn><obj><p3><nt><sg># away$

              |___|                                             |_____|
                |                                                   |
             LEMMA HEAD                                        LEMMA QUEUE

```

Chunks

See also: Chunking

```
^Verbcj<SV><vblex><ifi><p3><sg>{^come<vblex><ifi><p3><sg>$}$ ^pr<PREP>{^to<pr>$}$ ^det_nom<SN><f><sg>{^the<det><def><3>$ ^beach<n><3>$}$

   |   |______________________||__________________________|                                                          |
 CHUNK      CHUNK TAGS              LEXICAL UNITS IN                                                               LINKED
  NAME                                  THE CHUNK                                                                   TAG

   |________________________________________|
                       |
                     CHUNK


^det_nom<SN><f><sg>{^the<det><def><3>$ ^beach<n><3>$}$

                                   |______________|
                                          |
                                POINTERS TO CHUNK TAGS
        <1> <2> <3>
```

See also

- List of symbols
- Meaning of symbols * @ and dieze after a translation
- apertium-cleanstream which lets you avoid ad-hoc bash oneliners to get one word per line

Retrieved from "https://wiki.apertium.org/w/index.php?title=Apertium_stream_format&oldid=74007"

This page was last edited on 29 March 2022, at 07:12.

This page has been accessed 52,757 times.