

BNFT

About BNFT

BNFT is a tool for extraction information from text, checking against a BNF grammar (Backus-Naur-Form) and transform text into another text.

BNFT is pronounced B' N' F' it, or Benefit and the abbreviation stands for Backus Naur Form Transformer. This document is not a tutorial or BNF or EBNF and familiarity with these subject are assumed.

Running BNFT from the command line

BNFT is currently written in java, but it is the plan to port it to different environments. Currently you run it with

```
java -jar bnft.jar <BNFTfile> <-i inputfile> <-o outputfile>
```

Example:

```
java -jar bnft.jar Example.bnft -i Source.src
```

In the example no output file is specified and out is thus directed to the console.

The .BNFT file

The .BNFT file holds the transformation code, written in a modified EBNF form.

Examples of the .BNFT file is in the package

Using BNFT

BNFT is a simple style parser that uses some shortcuts to enable its function. The rules are:

Literals

A literal is enclosed in quotation marks “ or ‘. The double quotation mark means case sensitive, whereas the single quotation mark means case insensitive. Case insensitivity can be specified in the grammar itself and is only included as a convenience. Literals only advance the filepointer when the literal matches i.e. “Hello” will require “Hello” to be present at the current point in file.

Nonterminals

Nonterminals are the variables of the BNF. They can be specified using the “:” notation and the “=” notation:

```
number = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

assignment:

```
identifier := value
```

Single Optional “[“ and “]”

A single optional part of the grammar is defined with the square brackets “[“ and “]”. BNFT will advance the filepointer if the optional expression inside it is true. This is regardless whether or not the following possibly non optional expression is true i.e. BNFT does not search the grammar for a correct interpretation.

Multiple Optional “{” and “}”

A multiple optional continues to advance the filepointer unless the first following nonoptional item is true. This was introduced to prevent infinite loops i.e. “\”” { any_char } “\””. The any_char would eat the terminator – so any_char would have to exempt the “\””.

Or expression “|”

Or expressions can be constructed by using the “|” operator between arguments. A nonterminal using the “:” notation allows Or expression to be put on separate lines without the “|” operator:

```
AB_Char = "a" | "b" is the same as
AB_Char:
    "a"
    "b"
```

And expression

And expressions are default when a sequence is specified: “a” “b” “c” is that same as “abc” and “a” foo “c” is only satisfied when all three are satisfied.

Transformations

The “->” operator signified a transformation. It can be used to replace with, swap identifiers and control indentation. On the right side of the “->” operator you can write literals, nonterminals and the indent command characters “+” and “!”

```
String = "Hello" person -> "Goodbye " person " and have a nice day"
```

“#indent” will insert spaces that matches the current indentation (controlled by “#block”)
“#block” signifies a indentation so that nonterminals called from this line will have 1 greater indentation.

```
statement:
    single_statement          -> ! single_statement
block:
    "{ " { statement } " }"   -> + "{ " statement " }"
```

The Syntax BNF for the .BNFT file

```
alpha_char:
    "A".. "Z"
    "a".. "z"

numeric_char:
    "0".. "9"

hexadecimal_char:
    numeric_char
    "A".. "F"
    "a".. "f"

alphanumeric_char:
    numeric_char
    alpha_char
```



```
{ "#include" {whitespace} literal }  
{ entry }
```

```
script:  
  { body }
```