```
Oniguruma Regular Expressions Version 5.9.1    2007/09/05

syntax: ONIG_SYNTAX_RUBY (default)


1. Syntax elements

  \       escape (enable or disable meta character meaning)
  |       alternation
  (...)   group
  [...]   character class


2. Characters

  \t           horizontal tab (0x09)
  \v           vertical tab   (0x0B)
  \n           newline        (0x0A)
  \r           return         (0x0D)
  \b           back space     (0x08)
  \f           form feed      (0x0C)
  \a           bell           (0x07)
  \e           escape         (0x1B)
  \nnn         octal char            (encoded byte value)
  \xHH         hexadecimal char      (encoded byte value)
  \x{7HHHHHHH} wide hexadecimal char (character code point value)
  \cx          control char          (character code point value)
  \C-x         control char          (character code point value)
  \M-x         meta  (x|0x80)        (character code point value)
  \M-\C-x      meta control char     (character code point value)

 (* \b is effective in character class [...] only)


3. Character types

  .        any character (except newline)

  \w       word character

           Not Unicode:
             alphanumeric, "_" and multibyte char.

           Unicode:
             General_Category -- (Letter|Mark|Number|Connector_Punctuation)

  \W       non word char

  \s       whitespace char

           Not Unicode:
             \t, \n, \v, \f, \r, \x20

           Unicode:
                0009, 000A, 000B, 000C, 000D, 0085(NEL),
                General_Category -- Line_Separator
                                 -- Paragraph_Separator
                                 -- Space_Separator

  \S       non whitespace char

  \d       decimal digit char

           Unicode: General_Category -- Decimal_Number

  \D       non decimal digit char

  \h       hexadecimal digit char   [0-9a-fA-F]

  \H       non hexadecimal digit char


  Character Property

    * \p{property-name}
    * \p{^property-name}    (negative)
    * \P{property-name}     (negative)

    property-name:

     + works on all encodings
       Alnum, Alpha, Blank, Cntrl, Digit, Graph, Lower,
       Print, Punct, Space, Upper, XDigit, Word, ASCII,

     + works on EUC_JP, Shift_JIS
       Hiragana, Katakana

     + works on UTF8, UTF16, UTF32
       Any, Assigned, C, Cc, Cf, Cn, Co, Cs, L, Ll, Lm, Lo, Lt, Lu,
       M, Mc, Me, Mn, N, Nd, Nl, No, P, Pc, Pd, Pe, Pf, Pi, Po, Ps,
       S, Sc, Sk, Sm, So, Z, Zl, Zp, Zs,
       Arabic, Armenian, Bengali, Bopomofo, Braille, Buginese,
       Buhid, Canadian_Aboriginal, Cherokee, Common, Coptic,
       Cypriot, Cyrillic, Deseret, Devanagari, Ethiopic, Georgian,
       Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul,
       Hanunoo, Hebrew, Hiragana, Inherited, Kannada, Katakana,
       Kharoshthi, Khmer, Lao, Latin, Limbu, Linear_B, Malayalam,
       Mongolian, Myanmar, New_Tai_Lue, Ogham, Old_Italic, Old_Persian,
       Oriya, Osmanya, Runic, Shavian, Sinhala, Syloti_Nagri, Syriac,
       Tagalog, Tagbanwa, Tai_Le, Tamil, Telugu, Thaana, Thai, Tibetan,
       Tifinagh, Ugaritic, Yi



4. Quantifier

  greedy

    ?       1 or 0 times
    *       0 or more times
    +       1 or more times
    {n,m}   at least n but not more than m times
    {n,}    at least n times
    {,n}    at least 0 but not more than n times ({0,n})
    {n}     n times

  reluctant

    ??      1 or 0 times
    *?      0 or more times
    +?      1 or more times
    {n,m}?  at least n but not more than m times
    {n,}?   at least n times
    {,n}?   at least 0 but not more than n times (== {0,n}?)

  possessive (greedy and does not backtrack after repeated)

    ?+      1 or 0 times
    *+      0 or more times
    ++      1 or more times

    ({n,m}+, {n,}+, {n}+ are possessive op. in ONIG_SYNTAX_JAVA only)

    ex. /a*+/ === /(?>a*)/


5. Anchors

  ^       beginning of the line
  $       end of the line
  \b      word boundary
  \B      not word boundary
  \A      beginning of string
  \Z      end of string, or before newline at the end
  \z      end of string
  \G      matching start position


6. Character class

  ^...    negative class (lowest precedence operator)
  x-y     range from x to y
  [...]   set (character class in character class)
  ..&&..  intersection (low precedence at the next of ^)

    ex. [a-w&&[^c-g]z] ==> ([a-w] AND ([^c-g] OR z)) ==> [abh-w]

  * If you want to use '[', '-', ']' as a normal character
    in a character class, you should escape these characters by '\'.


  POSIX bracket ([:xxxxx:], negate [:^xxxxx:])

    Not Unicode Case:
```

```
    alnum    alphabet or digit char
    alpha    alphabet
    ascii    code value: [0 - 127]
    blank    \t, \x20
    cntrl
    digit    0-9
    graph    include all of multibyte encoded characters
    lower
    print    include all of multibyte encoded characters
    punct
    space    \t, \n, \v, \f, \r, \x20
    upper
    xdigit   0-9, a-f, A-F
    word     alphanumeric, "_" and multibyte characters


  Unicode Case:

    alnum    Letter | Mark | Decimal_Number
    alpha    Letter | Mark
    ascii    0000 - 007F
    blank    Space_Separator | 0009
    cntrl    Control | Format | Unassigned | Private_Use | Surrogate
    digit    Decimal_Number
    graph    [[:^space:]] && ^Control && ^Unassigned && ^Surrogate
    lower    Lowercase_Letter
    print    [[:graph:]] | [[:space:]]
    punct    Connector_Punctuation | Dash_Punctuation | Close_Punctuation |
             Final_Punctuation | Initial_Punctuation | Other_Punctuation |
             Open_Punctuation
    space    Space_Separator | Line_Separator | Paragraph_Separator |
             0009 | 000A | 000B | 000C | 000D | 0085
    upper    Uppercase_Letter
    xdigit   0030 - 0039 | 0041 - 0046 | 0061 - 0066
             (0-9, a-f, A-F)
    word     Letter | Mark | Decimal_Number | Connector_Punctuation



7. Extended groups

  (?#...)            comment

  (?imx-imx)         option on/off
                         i: ignore case
                         m: multi-line (dot(.) match newline)
                         x: extended form
  (?imx-imx:subexp)  option on/off for subexp

  (?:subexp)         not captured group
  (subexp)           captured group

  (?=subexp)         look-ahead
  (?!subexp)         negative look-ahead
  (?<=subexp)        look-behind
  (?<!subexp)        negative look-behind

                     Subexp of look-behind must be fixed character length.
                     But different character length is allowed in top level
                     alternatives only.
                     ex. (?<=a|bc) is OK. (?<=aaa(?:b|cd)) is not allowed.

                     In negative-look-behind, captured group isn't allowed,
                     but shy group(?:) is allowed.

  (?>subexp)         atomic group
                     don't backtrack in subexp.

  (?<name>subexp), (?'name'subexp)
                     define named group
                     (All characters of the name must be a word character.)

                     Not only a name but a number is assigned like a captured
                     group.

                     Assigning the same name as two or more subexps is allowed.
                     In this case, a subexp call can not be performed although
                     the back reference is possible.


8. Back reference

  \n          back reference by group number (n >= 1)
  \k<n>       back reference by group number (n >= 1)
  \k'n'       back reference by group number (n >= 1)
  \k<-n>      back reference by relative group number (n >= 1)
  \k'-n'      back reference by relative group number (n >= 1)
  \k<name>    back reference by group name
  \k'name'    back reference by group name

  In the back reference by the multiplex definition name,
  a subexp with a large number is referred to preferentially.
  (When not matched, a group of the small number is referred to.)

  * Back reference by group number is forbidden if named group is defined
    in the pattern and ONIG_OPTION_CAPTURE_GROUP is not setted.


  back reference with nest level

    level: 0, 1, 2, ...

    \k<n+level>     (n >= 1)
    \k<n-level>     (n >= 1)
    \k'n+level'     (n >= 1)
    \k'n-level'     (n >= 1)

    \k<name+level>
    \k<name-level>
```

```
      \k'name+level'
      \k'name-level'

      Destinate relative nest level from back reference position.

      ex 1.

        /\A(?<a>|.|(?:(?<b>.)\g<a>\k<b+0>))\z/.match("reer")

      ex 2.

        r = Regexp.compile(<<'__REGEXP__'.strip, Regexp::EXTENDED)
        (?<element> \g<stag> \g<content>* \g<etag> ){0}
        (?<stag> < \g<name> \s* > ){0}
        (?<name> [a-zA-Z_:]+ ){0}
        (?<content> [^<&]+ (\g<element> | [^<&]+)* ){0}
        (?<etag> </ \k<name+1> >){0}
        \g<element>
        __REGEXP__

        p r.match('<foo>f<bar>bbb</bar>f</foo>').captures



  9. Subexp call ("Tanaka Akira special")

    \g<name>    call by group name
    \g'name'    call by group name
    \g<n>       call by group number (n >= 1)
    \g'n'       call by group number (n >= 1)
    \g<-n>      call by relative group number (n >= 1)
    \g'-n'      call by relative group number (n >= 1)

    * left-most recursive call is not allowed.
       ex. (?<name>a|\g<name>b)   => error
           (?<name>a|b\g<name>c)  => OK

    * Call by group number is forbidden if named group is defined in the pattern
      and ONIG_OPTION_CAPTURE_GROUP is not setted.

    * If the option status of called group is different from calling position
      then the group's option is effective.

      ex. (?-i:\g<name>)(?i:(?<name>a)){0}  match to "A"


  10. Captured group

    Behavior of the no-named group (...) changes with the following conditions.
    (But named group is not changed.)

    case 1. /.../     (named group is not used, no option)

       (...) is treated as a captured group.
    case 2. /.../g    (named group is not used, 'g' option)

       (...) is treated as a no-captured group (?:...).

    case 3. /..(?<name>..)../   (named group is used, no option)

       (...) is treated as a no-captured group (?:...).
       numbered-backref/call is not allowed.

    case 4. /..(?<name>..)../G  (named group is used, 'G' option)

       (...) is treated as a captured group.
       numbered-backref/call is allowed.

    where
      g: ONIG_OPTION_DONT_CAPTURE_GROUP
      G: ONIG_OPTION_CAPTURE_GROUP

    ('g' and 'G' options are argued in ruby-dev ML)



  -----------------------------
  A-1. Syntax depend options

     + ONIG_SYNTAX_RUBY
       (?m): dot(.) match newline

     + ONIG_SYNTAX_PERL and ONIG_SYNTAX_JAVA
       (?s): dot(.) match newline
       (?m): ^ match after newline, $ match before newline


  A-2. Original extensions

     + hexadecimal digit char type  \h, \H
     + named group                  (?<name>...), (?'name'...)
     + named backref                \k<name>
     + subexp call                  \g<name>, \g<group-num>


  A-3. Lacked features compare with perl 5.8.0

     + \N{name}
     + \l,\u,\L,\U, \X, \C
     + (?{code})
     + (??{code})
     + (?(condition)yes-pat|no-pat)

     * \Q...\E
       This is effective on ONIG_SYNTAX_PERL and ONIG_SYNTAX_JAVA.


  A-4. Differences with Japanized GNU regex(version 0.12) of Ruby 1.8
```

```
    + add character property (\p{property}, \P{property})
    + add hexadecimal digit char type (\h, \H)
    + add look-behind
      (?<=fixed-char-length-pattern), (?<!fixed-char-length-pattern)
    + add possessive quantifier. ?+, *+, ++
    + add operations in character class. [], &&
      ('[' must be escaped as an usual char in character class.)
    + add named group and subexp call.
    + octal or hexadecimal number sequence can be treated as
      a multibyte code char in character class if multibyte encoding
      is specified.
      (ex. [\xa1\xa2], [\xa1\xa7-\xa4\xa1])
    + allow the range of single byte char and multibyte char in character
      class.
      ex. /[a-<<any EUC-JP character>>]/ in EUC-JP encoding.
    + effect range of isolated option is to next ')'.
      ex. (?:(?i)a|b) is interpreted as (?:(?i:a|b)), not (?:(?i:a)|b).
    + isolated option is not transparent to previous pattern.
      ex. a(?i)* is a syntax error pattern.
    + allowed incompleted left brace as an usual string.
      ex. /{/, /({)/, /a{2,3/ etc...
    + negative POSIX bracket [:^xxxx:] is supported.
    + POSIX bracket [:ascii:] is added.
    + repeat of look-ahead is not allowed.
      ex. /(?=a)*/, /(?!b){5}/
    + Ignore case option is effective to numbered character.
      ex. /\x61/i =~ "A"
    + In the range quantifier, the number of the minimum is omissible.
      /a{,n}/ == /a{0,n}/
      The simultanious abbreviation of the number of times of the minimum
      and the maximum is not allowed. (/a{,}/)
    + /a{n}?/ is not a non-greedy operator.
      /a{n}?/ == /(?:a{n})?/
    + invalid back reference is checked and cause error.
      /\1/, /(a)\2/
    + Zero-length match in infinite repeat stops the repeat,
      then changes of the capture group status are checked as stop condition.
      /(?:()|())*\1\2/ =~ ""
      /(?:\1a|())*/ =~ "a"


A-5. Disabled functions by default syntax

    + capture history

      (?@...) and (?@<name>...)

      ex. /(?@a)*/.match("aaa") ==> [<0-1>, <1-2>, <2-3>]

      see sample/listcap.c file.


A-6. Problems

    + Invalid encoding byte sequence is not checked.
```

```
     ex. UTF-8

     * Invalid first byte is treated as a character.
       /./u =~ "\xa3"

     * Incomplete byte sequence is not checked.
       /\w+/ =~ "a\xf3\x8ec"

// END
```