

# Contents

## 1 Module Pcre : Perl Compatibility Regular Expressions

1

## 1 Module Pcre : Perl Compatibility Regular Expressions

Exceptions

type error =

| Partial

String only matched the pattern partially

| BadPartial

Pattern contains items that cannot be used together with partial matching.

| BadPattern of string \* int

BadPattern (msg, pos) regular expression is malformed. The reason is in msg, the position of the error in the pattern in pos.

| BadUTF8

UTF8 string being matched is invalid

| BadUTF8Offset

Gets raised when a UTF8 string being matched with offset is invalid.

| MatchLimit

Maximum allowed number of match attempts with backtracking or recursion is reached during matching. ALL FUNCTIONS CALLING THE MATCHING ENGINE MAY RAISE IT!!!

| RecursionLimit

| InternalError of string

InternalError msg C-library exhibits unknown/undefined behaviour. The reason is in msg.

exception Error of error

Exception indicating PCRE errors.

exception Backtrack

Backtrack used in callout functions to force backtracking.

exception Regexp\_or of string \* error

Regexp\_or (pat, error) gets raised for sub-pattern pat by regexp\_or if it failed to compile.

Compilation and runtime flags and their conversion functions

type icflag

Internal representation of compilation flags

`type irflag`

Internal representation of runtime flags

`type cflag = [ 'ANCHORED`

`| 'AUTO_CALLOUT`

`| 'CASELESS`

`| 'DOLLAR_ENDONLY`

`| 'DOTALL`

`| 'EXTENDED`

`| 'EXTRA`

`| 'FIRSTLINE`

`| 'MULTILINE`

`| 'NO_AUTO_CAPTURE`

`| 'NO_UTF8_CHECK`

`| 'UNGREEDY`

`| 'UTF8 ]`

Compilation flags

`val cflags : cflag list -> icflag`

`cflags cflag_list` converts a list of compilation flags to their internal representation.

`val cflag_list : icflag -> cflag list`

`cflag_list cflags` converts internal representation of compilation flags to a list.

`type rflag = [ 'ANCHORED | 'NOTBOL | 'NOTEMPTY | 'NOTEOL | 'PARTIAL ]`

Runtime flags

`val rflags : rflag list -> irflag`

`rflags rflag_list` converts a list of runtime flags to their internal representation.

`val rflag_list : irflag -> rflag list`

`rflag_list rflags` converts internal representation of runtime flags to a list.

Information on the PCRE-configuration (build-time options)

`val version : string`

Version information

Version of the PCRE-C-library

`val config_utf8 : bool`

Indicates whether UTF8-support is enabled

`val config_newline : char`

Character used as newline

```

val config_link_size : int
    Number of bytes used for internal linkage of regular expressions

val config_match_limit : int
    Default limit for calls to internal matching function

val config_match_limit_recursion : int
    Default limit recursion for calls to internal matching function

val config_stackrecurse : bool
    Indicates use of stack recursion in matching function

    Information on patterns
type firstbyte_info = [ 'ANCHORED | 'Char of char | 'Start_only ]
    Information on matching of "first chars" in patterns

type study_stat = [ 'Not_studied | 'Optimal | 'Studied ]
    Information on the study status of patterns

type regexp
    Compiled regular expressions

val options : regexp -> icflag
    options regexp
    Returns compilation flags of regexp.

val size : regexp -> int
    size regexp
    Returns memory size of regexp.

val studysize : regexp -> int
    studysize regexp
    Returns memory size of study information of regexp.

val capturecount : regexp -> int
    capturecount regexp
    Returns number of capturing subpatterns in regexp.

val backrefmax : regexp -> int
    backrefmax regexp
    Returns number of highest backreference in regexp.

val namecount : regexp -> int

```

`namecount regexp`

**Returns** number of named subpatterns in `regexp`.

`val names : regexp -> string array`

`names regex`

**Returns** array of names of named substrings in `regexp`.

`val nameentrysize : regexp -> int`

`nameentrysize regexp`

**Returns** size of longest name of named subpatterns in `regexp` + 3.

`val firstbyte : regexp -> firstbyte_info`

`firstbyte regexp`

**Returns** firstbyte info on `regexp`.

`val firsttable : regexp -> string option`

`firsttable regexp`

**Returns** some 256-bit (32-byte) fixed set table in form of a string for `regexp` if available, `None` otherwise.

`val lastliteral : regexp -> char option`

`lastliteral regexp`

**Returns** some last matching character of `regexp` if available, `None` otherwise.

`val study_stat : regexp -> study_stat`

`study_stat regexp`

**Returns** study status of `regexp`.

`val get_stringnumber : regexp -> string -> int`

`get_stringnumber rex name`

**Raises** `Invalid_arg` if there is no such named substring.

**Returns** the index of the named substring `name` in regular expression `rex`. This index can then be used with `get_substring`.

`val get_match_limit : regexp -> int option`

`get_match_limit rex`

**Returns** some match limit of regular expression `rex` or `None`.

Compilation of patterns

`type chtables`

Alternative set of char tables for pattern matching

`val maketables : unit -> chtables`

Generates new set of char tables for the current locale.

`val regexp :`

`?study:bool ->`

`?limit:int ->`

`?limit_recursion:int ->`

`?iflags:icflag ->`

`?flags:cflag list -> ? chtables: chtables -> string -> regexp`

`regexp ?study ?limit ?limit_recursion ?iflags ?flags ? chtables pattern` compiles `pattern` with `flags` when given, with `iflags` otherwise, and with char tables  `chtables`. If `study` is true, then the resulting regular expression will be studied. If `limit` is specified, this sets a limit to the amount of recursion and backtracking (only lower than the builtin default!). If this limit is exceeded, `MatchLimit` will be raised during matching.

**Returns** the regular expression.

For detailed documentation on how you can specify PERL-style regular expressions (= patterns), please consult the PCRE-documentation ("man pcrepattern") or PERL-manuals.

**See also** [www.perl.com](http://www.perl.com)[<http://www.perl.com>]

`val regexp_or :`

`?study:bool ->`

`?limit:int ->`

`?limit_recursion:int ->`

`?iflags:icflag ->`

`?flags:cflag list ->`

`? chtables: chtables -> string list -> regexp`

`regexp_or ?study ?limit ?limit_recursion ?iflags ?flags ? chtables patterns` like `Pcre.regexp[1]`, but combines `patterns` as alternatives (or-patterns) into one regular expression.

`val quote : string -> string`

`quote str`

**Returns** the quoted string of `str`.

Subpattern extraction

`type substrings`

Information on substrings after pattern matching

`val get_subject : substrings -> string`

`get_subject substrings`

**Returns** the subject string of `substrings`.

`val num_of_subs : substrings -> int`

`num_of_subs substrings`

**Returns** number of strings in `substrings` (whole match inclusive).

`val get_substring : substrings -> int -> string`

`get_substring substrings n`

**Raises**

- `Invalid_argument` if `n` is not in the range of the number of substrings.
- `Not_found` if the corresponding subpattern did not capture a substring.

**Returns** the `n`th substring (0 is whole match) of `substrings`.

`val get_substring_ofs : substrings -> int -> int * int`

`get_substring_ofs substrings n`

**Raises**

- `Invalid_argument` if `n` is not in the range of the number of substrings.
- `Not_found` if the corresponding subpattern did not capture a substring.

**Returns** the offset tuple of the `n`th substring of `substrings` (0 is whole match).

`val get_substrings : ?full_match:bool -> substrings -> string array`

`get_substrings ?full_match substrings`

**Returns** the array of substrings in `substrings`. It includes the full match at index 0 when `full_match` is `true`, the captured substrings only when it is `false`. If a subpattern did not capture a substring, the empty string is returned in the corresponding position instead.

`val get_opt_substrings :`

`?full_match:bool -> substrings -> string option array`

`get_opt_substrings ?full_match substrings`

**Returns** the array of optional substrings in `substrings`. It includes `Some full_match_str` at index 0 when `full_match` is `true`, `Some captured_substrings` only when it is `false`. If a subpattern did not capture a substring, `None` is returned in the corresponding position instead.

`val get_named_substring : regexp -> string -> substrings -> string`

`get_named_substring rex name substrings`

**Raises**

- `Invalid_argument` if there is no such named substring.
- `Not_found` if the corresponding subpattern did not capture a substring.

**Returns** the named substring `name` in regular expression `rex` and `substrings`.

`val get_named_substring_ofs : regexp -> string -> substrings -> int * int`

`get_named_substring_ofs rex name substrings`

**Raises**

- `Invalid_argument` if there is no such named substring.
- `Not_found` if the corresponding subpattern did not capture a substring.

**Returns** the offset tuple of the named substring `name` in regular expression `rex` and `substrings`.

Callouts

```
type callout_data = {  
  callout_number : int ;  
    Callout number  
  
  substrings : substrings ;  
    Substrings matched so far  
  
  start_match : int ;  
    Subject start offset of current match attempt  
  
  current_position : int ;  
    Subject offset of current match pointer  
  
  capture_top : int ;  
    Number of the highest captured substring so far  
  
  capture_last : int ;  
    Number of the most recently captured substring  
  
  pattern_position : int ;  
    Offset of next match item in pattern string  
  
  next_item_length : int ;  
    Length of next match item in pattern string  
}
```

```
type callout = callout_data -> unit
```

Type of callout functions

Callouts are referred to in patterns as "(?Cn)" where "n" is a `callout_number` ranging from 0 to 255. Substrings captured so far are accessible as usual via `substrings`. You will have to consider `capture_top` and `capture_last` to know about the current state of valid substrings.

By raising exception `Backtrack` within a callout function, the user can force the pattern matching engine to backtrack to other possible solutions. Other exceptions will terminate matching immediately and return control to OCaml.

Matching of patterns and subpattern extraction

```
val pcre_exec :  
  ?iflags:irflag ->
```

```

?flags:rflag list ->
?rex:regexp ->
?pat:string -> ?pos:int -> ?callout:callout -> string -> int array
  pcre_exec ?iflags ?flags ?rex ?pat ?pos ?callout subj

```

**Raises** `Not_found` if pattern does not match.

**Returns** an array of offsets that describe the position of matched subpatterns in the string `subj` starting at position `pos` with pattern `pat` when given, regular expression `rex` otherwise. The array also contains additional workspace needed by the match engine. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

```

val exec :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string -> ?pos:int -> ?callout:callout -> string -> substrings
    exec ?iflags ?flags ?rex ?pat ?pos ?callout subj

```

**Raises** `Not_found` if pattern does not match.

**Returns** substring information on string `subj` starting at position `pos` with pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

```

val exec_all :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int -> ?callout:callout -> string -> substrings array
    exec_all ?iflags ?flags ?rex ?pat ?pos ?callout subj

```

**Raises** `Not_found` if pattern does not match.

**Returns** an array of substring information of all matching substrings in string `subj` starting at position `pos` with pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

```

val next_match :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int -> ?callout:callout -> substrings -> substrings
    next_match ?iflags ?flags ?rex ?pat ?pos ?callout subtrs

```

**Raises**

- `Not_found` if pattern does not match.



- `Invalid_arg` if `pos` let matching start outside of the subject string.

**Returns** substring information on the match that follows on the last match denoted by `substrs`, jumping over `pos` characters (also backwards!), using pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

```
val extract :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?full_match:bool -> ?callout:callout -> string -> string array
  extract ?iflags ?flags ?rex ?pat ?pos ?full_match ?callout subj
```

**Raises** `Not_found` if pattern does not match.

**Returns** the array of substrings that match `subj` starting at position `pos`, using pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. It includes the full match at index 0 when `full_match` is `true`, the captured substrings only when it is `false`. Callouts are handled by `callout`. If a subpattern did not capture a substring, the empty string is returned in the corresponding position instead.

```
val extract_opt :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?full_match:bool -> ?callout:callout -> string -> string option array
  extract_opt ?iflags ?flags ?rex ?pat ?pos ?full_match ?callout subj
```

**Raises** `Not_found` if pattern does not match.

**Returns** the array of optional substrings that match `subj` starting at position `pos`, using pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. It includes `Some full_match_str` at index 0 when `full_match` is `true`, `Some captured-substrings` only when it is `false`. Callouts are handled by `callout`. If a subpattern did not capture a substring, `None` is returned in the corresponding position instead.

```
val extract_all :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
```

```
?full_match:bool -> ?callout:callout -> string -> string array array
  extract_all ?iflags ?flags ?rex ?pat ?pos ?full_match ?callout subj
Raises Not_found if pattern does not match.
```

**Returns** an array of arrays of all matching substrings that match `subj` starting at position `pos`, using pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. It includes the full match at index 0 of the extracted string arrays when `full_match` is `true`, the captured substrings only when it is `false`. Callouts are handled by `callout`.

```
val extract_all_opt :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?full_match:bool ->
  ?callout:callout -> string -> string option array array
  extract_all_opt ?iflags ?flags ?rex ?pat ?pos ?full_match ?callout subj
Raises Not_found if pattern does not match.
```

**Returns** an array of arrays of all optional matching substrings that match `subj` starting at position `pos`, using pattern `pat` when given, regular expression `rex` otherwise. Uses `flags` when given, the precompiled `iflags` otherwise. It includes `Some full_match_str` at index 0 of the extracted string arrays when `full_match` is `true`, `Some captured_substrings` only when it is `false`. Callouts are handled by `callout`. If a subpattern did not capture a substring, `None` is returned in the corresponding position instead.

```
val pmatch :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string -> ?pos:int -> ?callout:callout -> string -> bool
  pmatch ?iflags ?flags ?rex ?pat ?pos ?callout subj
```

**Returns** `true` if `subj` is matched by pattern `pat` when given, regular expression `rex` otherwise, starting at position `pos`. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

String substitution

```
type substitution
```

Information on substitution patterns

```
val subst : string -> substitution
  subst str converts the string str representing a substitution pattern to the internal
  representation
```

The contents of the substitution string `str` can be normal text mixed with any of the following (mostly as in PERL):

- `$/0-9/+` - a "\$" immediately followed by an arbitrary number. "\$0" stands for the name of the executable, any other number for the n-th backreference.
- `$$` - the whole matched pattern
- `$'` - the text before the match
- `$'` - the text after the match
- `$+` - the last group that matched
- `$$` - a single "\$"
- `$!` - delimiter which does not appear in the substitution. Can be used to part "\$0-9+" from an immediately following other number.

```
val replace :  
  ?iflags:irflag ->  
  ?flags:rflag list ->  
  ?rex:regexp ->  
  ?pat:string ->  
  ?pos:int ->  
  ?itempl:substitution ->  
  ?templ:string -> ?callout:callout -> string -> string  
  replace ?iflags ?flags ?rex ?pat ?pos ?itempl ?templ ?callout subj replaces all  
  substrings of subj matching pattern pat when given, regular expression rex otherwise,  
  starting at position pos with the substitution string templ when given, itempl otherwise.  
  Uses flags when given, the precompiled iflags otherwise. Callouts are handled by  
  callout.
```

**Raises Failure** if there are backreferences to nonexistent subpatterns.

```
val qreplace :  
  ?iflags:irflag ->  
  ?flags:rflag list ->  
  ?rex:regexp ->  
  ?pat:string ->  
  ?pos:int -> ?templ:string -> ?callout:callout -> string -> string  
  qreplace ?iflags ?flags ?rex ?pat ?pos ?templ ?callout subj replaces all substrings  
  of subj matching pattern pat when given, regular expression rex otherwise, starting at  
  position pos with the string templ. Uses flags when given, the precompiled iflags  
  otherwise. Callouts are handled by callout.
```

```
val substitute_substrings :  
  ?iflags:irflag ->  
  ?flags:rflag list ->  
  ?rex:regexp ->
```

```

?pat:string ->
?pos:int ->
?callout:callout ->
subst:(substrings -> string) -> string -> string
    substitute_substrings ?iflags ?flags ?rex ?pat ?pos ?callout ~subst subj
    replaces all substrings of subj matching pattern pat when given, regular expression rex
    otherwise, starting at position pos with the result of function subst applied to the
    substrings of the match. Uses flags when given, the precompiled iflags otherwise.
    Callouts are handled by callout.

```

```

val substitute :
    ?iflags:irflag ->
    ?flags:rflag list ->
    ?rex:regexp ->
    ?pat:string ->
    ?pos:int ->
    ?callout:callout -> subst:(string -> string) -> string -> string
    substitute ?iflags ?flags ?rex ?pat ?pos ?callout ~subst subj replaces all
    substrings of subj matching pattern pat when given, regular expression rex otherwise,
    starting at position pos with the result of function subst applied to the match. Uses flags
    when given, the precompiled iflags otherwise. Callouts are handled by callout.

```

```

val replace_first :
    ?iflags:irflag ->
    ?flags:rflag list ->
    ?rex:regexp ->
    ?pat:string ->
    ?pos:int ->
    ?itempl:substitution ->
    ?templ:string -> ?callout:callout -> string -> string
    replace_first ?iflags ?flags ?rex ?pat ?pos ?itempl ?templ ?callout subj
    replaces the first substring of subj matching pattern pat when given, regular expression rex
    otherwise, starting at position pos with the substitution string templ when given, itempl
    otherwise. Uses flags when given, the precompiled iflags otherwise. Callouts are handled
    by callout.

```

**Raises Failure** if there are backreferences to nonexistent subpatterns.

```

val qreplace_first :
    ?iflags:irflag ->
    ?flags:rflag list ->
    ?rex:regexp ->
    ?pat:string ->
    ?pos:int -> ?templ:string -> ?callout:callout -> string -> string
    qreplace_first ?iflags ?flags ?rex ?pat ?pos ?templ ?callout subj replaces the
    first substring of subj matching pattern pat when given, regular expression rex otherwise,

```

starting at position `pos` with the string `templ`. Uses `flags` when given, the precompiled `iflags` otherwise. Callouts are handled by `callout`.

```
val substitute_substrings_first :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?callout:callout ->
  subst:(substrings -> string) -> string -> string
  substitute_substrings_first ?iflags ?flags ?rex ?pat ?pos ?callout ~subst subj
  replaces the first substring of subj matching pattern pat when given, regular expression rex
  otherwise, starting at position pos with the result of function subst applied to the
  substrings of the match. Uses flags when given, the precompiled iflags otherwise.
  Callouts are handled by callout.
```

```
val substitute_first :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?callout:callout -> subst:(string -> string) -> string -> string
  substitute_first ?iflags ?flags ?rex ?pat ?pos ?callout ~subst subj replaces the
  first substring of subj matching pattern pat when given, regular expression rex otherwise,
  starting at position pos with the result of function subst applied to the match. Uses flags
  when given, the precompiled iflags otherwise. Callouts are handled by callout.
```

### Splitting

```
val split :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int -> ?max:int -> ?callout:callout -> string -> string list
  split ?iflags ?flags ?rex ?pat ?pos ?max ?callout subj splits subj into a list of at
  most max strings, using as delimiter pattern pat when given, regular expression rex
  otherwise, starting at position pos. Uses flags when given, the precompiled iflags
  otherwise. If max is zero, trailing empty fields are stripped. If it is negative, it is treated as
  arbitrarily large. If neither pat nor rex are specified, leading whitespace will be stripped!
  Should behave exactly as in PERL. Callouts are handled by callout.
```

```
val asplit :
  ?iflags:irflag ->
  ?flags:rflag list ->
```

`?rex:regexp ->`  
`?pat:string ->`  
`?pos:int -> ?max:int -> ?callout:callout -> string -> string array`  
`asplit ?iflags ?flags ?rex ?pat ?pos ?max ?callout subj` same as `Pcre.split[1]` but  
**Returns** an array instead of a list.

```

type split_result =
  | Text of string
      Text part of splitted string
  | Delim of string
      Delimiter part of splitted string
  | Group of int * string
      Subgroup of matched delimiter (subgroup_nr, subgroup_str)
  | NoGroup
      Unmatched subgroup
      Result of a Pcre.full_split[1]
  
```

```

val full_split :
  ?iflags:irflag ->
  ?flags:rflag list ->
  ?rex:regexp ->
  ?pat:string ->
  ?pos:int ->
  ?max:int -> ?callout:callout -> string -> split_result list
  
```

`full_split ?iflags ?flags ?rex ?pat ?pos ?max ?callout subj` splits `subj` into a list of at most `max` elements of type `"split_result"`, using as delimiter pattern `pat` when given, regular expression `rex` otherwise, starting at position `pos`. Uses `flags` when given, the precompiled `iflags` otherwise. If `max` is zero, trailing empty fields are stripped. If it is negative, it is treated as arbitrarily large. Should behave exactly as in PERL. Callouts are handled by `callout`.

Additional convenience functions

```

val foreach_line : ?ic:Pervasives.in_channel -> (string -> unit) -> unit
  
```

`foreach_line ?ic f` applies `f` to each line in inchannel `ic` until the end-of-file is reached.

```

val foreach_file :
  string list -> (string -> Pervasives.in_channel -> unit) -> unit
  
```

`foreach_file filenames f` opens each file in the list `filenames` for input and applies `f` to each filename and the corresponding channel. Channels are closed after each operation (even when exceptions occur - they get reraised afterwards!).

**UNSAFE STUFF - USE WITH CAUTION!**

```

val unsafe_pcre_exec :
  
```

irflag ->

regex ->

int -> string -> int -> int array -> callout option -> unit

`unsafe_pcre_exec` flags rex pos subject subgroup\_offsets offset\_vector. You should read the C-source to know what happens. If you do not understand it - **don't use this function!**

val make\_ovector : regex -> int \* int array

`make_ovector` regex calculates the tuple (subgroups2, ovector) which is the number of subgroup offsets and the offset array.