

KOKUM

Regular Expressions & Natural Patterns

A practical, verified programming manual

`^Kokum(?:\s+Natural)?\s+Patterns$`

Product	Kokum Programming Language
Version	0.29.0 · Phase 4.16.7
Edition	September 2026
Scope	Direct RegEx API, compiled REGEX resources, FILE subjects, and Natural Pattern operations

Prepared for developers, trainers, testers, and application authors

Document profile

This manual documents the regular-expression facilities present in **Kokum 0.29.0 Phase 4.16.7**. It covers the compact direct API, reusable compiled **REGEX** resources, ordinary **FILE** subjects, and Kokum's readable Natural Pattern statements.

Item	Value
Language baseline	Kokum 0.29.0 Phase 4.16.7
RegEx backend	Native PCRE2 8-bit API; UTF-8 and Unicode properties are enabled by default
Natural operations	FIND, REPLACE, SPLIT, VALIDATE, ITERATE, and CAPTURE
Execution paths	Interpreter, canonical IR, bytecode execution, and external KokumVM
Example status	22 standalone examples checked, compiled, verified, and executed through kokumc and kokumvm

IMPORTANT

Version scope: Syntax and behavior in this manual are tied to the Phase 4.16.7 implementation. Earlier Phase 4.16 packages may not contain natural **REPLACE**, **SPLIT**, **VALIDATE**, **CAPTURE**, or **ITERATE**.

All complete programs in the manual use **FUNCTION Main AS INTEGER** and the **?** print statement. Output shown under a program is the observed output from the verified source tree. File examples list the required fixture contents immediately before the program.

Contents

Section	Title	Purpose
1	Understanding Kokum pattern matching	The two interfaces and how to choose between them
2	Running the examples	Check, run, compile, verify, and execute
3	Direct RegEx API	Seven public functions and their return values
4	Writing raw regular expressions	Syntax, escaping, groups, Unicode, flags, and greediness
5	Compiled REGEX resources	Compile once, reuse, inspect, and close
6	RegEx with FILE subjects	Current-position semantics and ownership
7	Natural Pattern fundamentals	Sources, selectors, units, predicates, atoms, and quantifiers
8	FIND statements	Collect, count, stream, limit, and select
9	Captures and detailed results	Groups, names, positions, and result maps
10	Natural REPLACE	Readable transformations and capture references
11	Natural SPLIT	Delimiter control, limits, and empty fields
12	VALIDATE	Whole-source validation and Boolean conditions
13	ITERATE	Execute ordinary Kokum code for each match
14	IDE assistance and Pattern Inspector	Completion, formatting, diagnostics, and previews
15	Diagnostics and troubleshooting	KRG errors and common corrections
16	Performance, safety, and portability	Streaming, limits, reuse, and engine parity
17	Practical recipes	Copy-ready patterns for common tasks
A–E	Appendices	Quick references, fields, diagnostics, and glossary

TIP

Navigation: In Microsoft Word or LibreOffice Writer, use the document heading/navigation pane to jump directly to any numbered chapter or appendix.

1. Understanding Kokum pattern matching

A **regular expression** describes text using a compact pattern language. The text being examined is the **subject**. A successful portion of the subject is a **match**. Parentheses or Natural Pattern **CAPTURE** declarations can retain parts of a match as **groups**.

Kokum provides two complementary interfaces. They share the same native RegEx runtime, but they serve different programming styles.

Two ways to use patterns in Kokum

Both routes share one runtime and produce portable behavior across all execution engines.

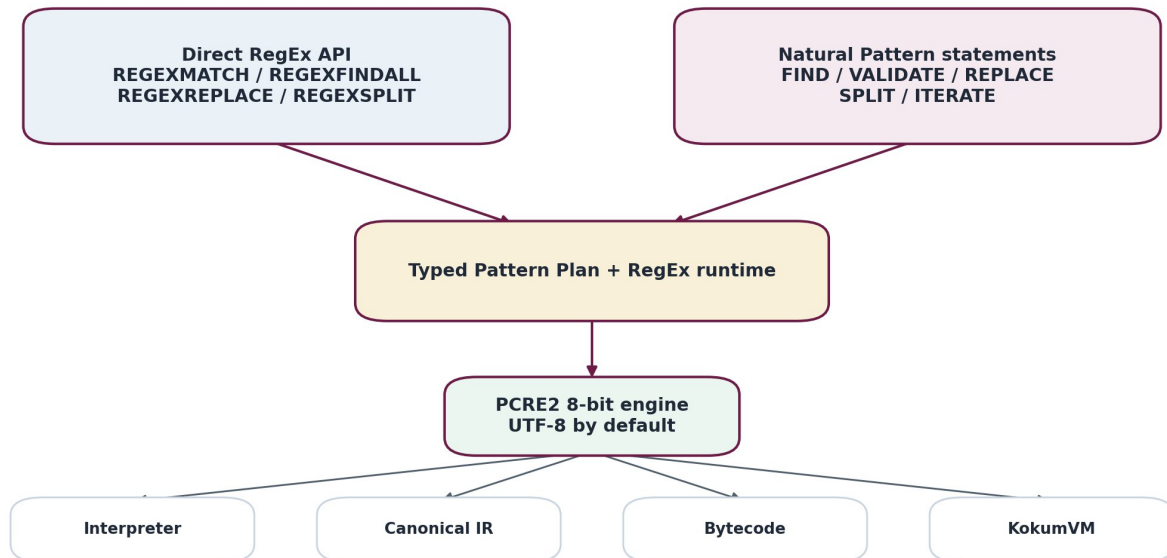


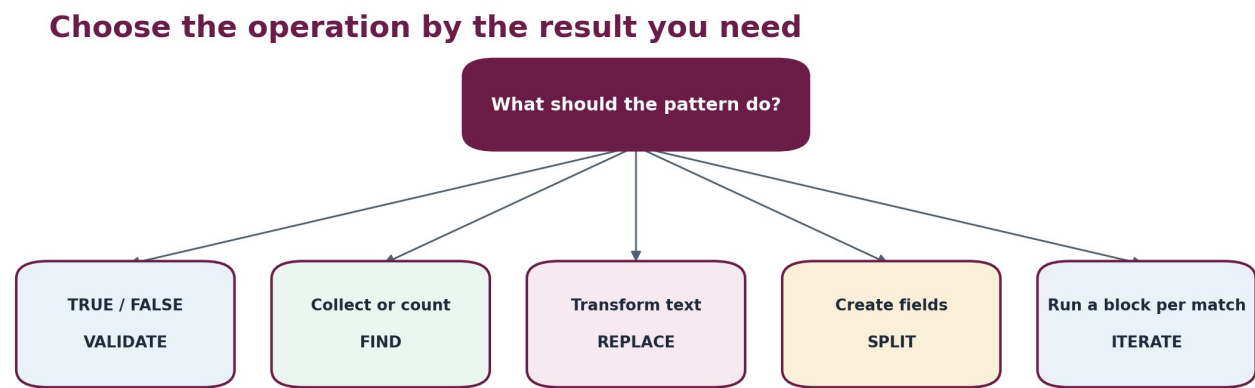
Figure 1. Direct RegEx and Natural Pattern share one native execution layer.

1.1 Direct RegEx API

Use the direct API when a standard PCRE2 pattern is the clearest representation, when a pattern already exists, or when a compact expression is desirable. Examples include `REGEXMATCH()`, `REGEXFINDALL()`, and `REGEXREPLACE()`.

1.2 Natural Pattern language

Use Natural Pattern statements when readability, FILE streaming, source selectors, detailed result maps, or IDE inspection are important. A statement such as `EXACTLY 2 LETTERS FOLLOWED BY EXACTLY 4 DIGITS` is parsed into a typed Pattern Plan; Kokum does not perform fragile word-for-word substitution.



Use direct REGEX... functions when you prefer compact PCRE2 syntax or already have a raw pattern.

Use Natural Pattern statements when readability, selectors, FILE streaming, details, or IDE inspection matter.

Figure 2. Select an operation according to the result required.

Need	Recommended form	Reason
One Boolean answer	VALIDATE	Whole-source validation is automatic for pattern sequences and raw RegEx leaves
First match or every match	REGEXFIND / REGEXFINDALL	Compact direct call
Readable line or word search	FIND	Selectors, streaming, INTO, COUNT TO, and details
Reusable pattern used many times	REGEXCOMPILE	Compile once and invoke methods repeatedly
Controlled match replacement	REPLACE FROM	Natural captures, selectors, and LIMIT
Delimiter-aware field creation	SPLIT FROM	KEEP DELIMITERS and delimiter LIMIT
Ordinary code for each match	ITERATE FROM	Provides a detailed match map in each loop iteration

2. Running the examples

Save a complete example as a `.kokum` source file. From the Kokum working directory, use the following commands. On Windows, use `kokumc.exe` and `kokumvm.exe`.

Build and run workflow

```
./kokumc check example.kokum
./kokumc run example.kokum
./kokumc compile example.kokum -o example.kbc
./kokumvm verify example.kbc
./kokumvm example.kbc
```

Command	Purpose
<code>kokumc check</code>	Parse and perform semantic checks without running
<code>kokumc run</code>	Execute through the source interpreter
<code>kokumc run-ir</code>	Execute the canonical IR
<code>kokumc run-bytecode</code>	Compile and execute using the bytecode reference path
<code>kokumc compile</code>	Write portable <code>.kbc</code> bytecode

Command	Purpose
kokumvm verify	Validate a bytecode file before execution
kokumvm file.kbc	Execute with the external KokumVM

2.1 First complete program

This program validates a two-letter, four-digit code. Anchors make the pattern cover the complete subject.

Verified Kokum example · 01_match.kokum

```
FUNCTION Main AS INTEGER
  LOCAL valid AS LOGICAL
  valid = REGEXMATCH("^([A-Z]{2}[0-9]{4})$", "AB1234")
  ? valid
  RETURN 0
ENDFUNC
```

Output

.T.

NOTE

Boolean display: Kokum prints logical true and false as **.T.** and **.F..**

3. Direct RegEx API

The direct API consists of seven append-only built-ins. A pattern argument may be either a **STRING** containing raw RegEx syntax or a compiled **REGEX** value.

Function	Returns	Behavior
REGEXCOMPILE(pattern [, flags])	REGEX	Compile a reusable native pattern resource
REGEXMATCH(patternOrRegex, subject [, flags])	LOGICAL	True when a match exists; anchor the pattern for full-subject validation
REGEXFIND(patternOrRegex, subject [, flags])	STRING or NULL	Return the first complete match
REGEXFINDALL(patternOrRegex, subject [, flags])	ARRAY	Return all non-overlapping complete matches
REGEXREPLACE(patternOrRegex, subject, replacement [, flags])	STRING	Replace every match and return a new string
REGEXSPLIT(patternOrRegex, subject [, flags])	ARRAY	Split around every delimiter match; delimiters are not returned
REGEXESCAPE(text)	STRING	Escape literal text so RegEx metacharacters lose their special meaning

IMPORTANT

Search versus validation: **REGEXMATCH()** reports whether the pattern matches somewhere in the subject. Use **^...\$**, **\A... \z**, or Natural **VALIDATE** when the complete subject must conform.

3.1 Find the first match and all matches

A no-match result from **REGEXFIND()** is **NULL**; **REGEXFINDALL()** always returns an array, which may be empty.

Verified Kokum example · 02_find.kokum

```
FUNCTION Main AS INTEGER
  LOCAL first
  LOCAL all AS ARRAY

  first = REGEXFIND("[0-9]+", "Order 45892 ready")
  all = REGEXFINDALL("[A-Z]{2}[0-9]+", "AA12 BB340 cc7")

  ? first
  ? JSONENCODE(all)
  RETURN 0
ENDFUNC
```

Output

```
45892
["AA12", "BB340"]
```

3.2 Replace and split

Direct replacement is global. Numbered capture references such as **\$1** and **\$2** are expanded in the replacement.

Verified Kokum example · 03_replace_split.kokum

```
FUNCTION Main AS INTEGER
  LOCAL parts AS ARRAY

  ? REGEXREPLACE("([A-Z]+)([0-9]+)", "AA12 BB35", "$1-$2")
  parts = REGEXSPLIT("[,;|]", "A,B;C|D")
  ? JSONENCODE(parts)
  RETURN 0
ENDFUNC
```

Output

```
AA-12 BB-35
["A", "B", "C", "D"]
```

3.3 Replacement references in the direct API

Reference	Meaning
\$0 or \$&	Complete match
\$1, \$2, ...	Numbered capture group
\${name}	Named capture created with (?<name>...)
\$\$	One literal dollar sign

3.4 Named captures in direct replacement

Named groups make replacements easier to read while numbered references remain available.

Verified Kokum example · 19_named_replace.kokum

```
FUNCTION Main AS INTEGER
  LOCAL pattern AS STRING
  LOCAL output AS STRING

  pattern = "(?<code>[A-Z]{2})(?<number>[0-9]+)"
  output = REGEXREPLACE(pattern, "AA12 BB3", "${code}:$2")
  ? output
  RETURN 0
ENDFUNC
```

Output

AA:12 BB:3

3.5 Safely search for literal user text

REGEXESCAPE() is the correct way to insert an arbitrary literal into a raw RegEx pattern.

Verified Kokum example · 04_escape.kokum

```
FUNCTION Main AS INTEGER
  LOCAL literal AS STRING
  LOCAL pattern AS STRING

  literal = "[file].txt"
  pattern = "^" + REGEXESCAPE(literal) + "$"

  ? REGEXESCAPE(literal)
  ? REGEXMATCH(pattern, "[file].txt")
  RETURN 0
ENDFUNC
```

Output

```
\[file\]\.txt
.T.
```

4. Writing raw regular expressions

Kokum uses the PCRE2 8-bit pattern engine behind a stable Kokum API. This chapter lists the most useful syntax. It is intentionally a practical subset rather than a complete PCRE2 language reference.

4.1 Literal characters and metacharacters

Syntax	Meaning	Example
abc	Literal text	abc
.	Any character except newline unless s is enabled	a.c
\.	A literal dot	file\.txt
[abc]	One listed character	[,;]
[^abc]	One character not listed	[^0-9]
[A-Z]	Character range	[A-Z]{2}
	Alternative	cat dog

4.2 Quantifiers

Syntax	Count	Example
?	Zero or one	colou?r
*	Zero or more	A*
+	One or more	[0-9]+
{n}	Exactly n	[A-Z]{2}
{n,}	At least n	[0-9]{4,}
{m,n}	Between m and n	[A-Za-z]{3,16}
? +? ??	Lazy version of a quantifier	<.?>

4.3 Anchors and boundaries

Syntax	Meaning
	Start of subject, or start of line with the <code>m</code> flag
	End of subject, or end of line with the <code>m</code> flag
<code>A</code>	Absolute start of subject
<code>Z</code>	Absolute end of subject
<code>b</code>	Word boundary
<code>R</code>	Unicode newline sequence

4.4 Groups, alternatives, and Unicode classes

Syntax	Meaning
<code>(...)</code>	Capturing group
<code>(?:...)</code>	Non-capturing group
<code>(?<name>...)</code>	Named capturing group
<code>\d \w \s</code>	Digit, word character, and whitespace shorthands
<code>\p{L}</code>	Unicode letter property
<code>\p{M}</code>	Unicode combining-mark property
<code>[\p{L}\p{M}]+</code>	A useful Unicode word pattern that retains combining marks

Two escaping layers in a Kokum source string



The Kokum string parser consumes one backslash. Use `REGEXESCAPE()` for user-supplied literal text.

Figure 3. A backslash must survive both the Kokum string parser and the RegEx parser.

4.5 Escaping inside Kokum strings

A raw RegEx backslash must normally be written as two backslashes inside a Kokum string. For example, the RegEx `\d+` is written as `"\\d+"` in source. Ordinary braces used by RegEx quantifiers do not need special Kokum escaping.

TIP

Literal values: When the pattern contains text supplied by a user, call `REGEXESCAPE()` instead of trying to add backslashes manually.

4.6 Unicode-aware matching

The default `u` mode supports Unicode properties. Indic words often contain combining marks, so `[\p{L}\p{M}]+` is usually more useful than `\p{L}+` alone.

Verified Kokum example · 05_unicode.kokum

```
FUNCTION Main AS INTEGER
  LOCAL words AS ARRAY

  ? REGEXMATCH("^\\p{L}+$", "कखग")
  words = REGEXFINDALL("[\\p{L}\\p{M}]+", "Kokum मराठी 29")
  ? JSONENCODE(words)
  RETURN 0
ENDFUNC
```

Output

```
.T.
["Kokum", "मराठी"]
```

4.7 Flags

Flag	Meaning	Practical effect
<code>u</code>	UTF-8 and Unicode properties; default	<code>\w</code> , properties, and case handling use Unicode semantics
<code>a</code>	ASCII character-property mode	Mutually exclusive with <code>u</code> ; UTF-8 boundaries remain safe
<code>i</code>	Case-insensitive	<code>apple</code> can match <code>APPLE</code>
<code>m</code>	Multiline anchors	<code>^</code> and <code>\$</code> also apply at line boundaries
<code>s</code>	Dot matches newline	<code>.</code> can cross a line terminator
<code>x</code>	Extended/free-spacing syntax	Unescaped spaces and comments can improve raw-pattern readability
<code>U</code>	Ungreedy quantifiers	Greedy quantifiers become lazy by default
<code>n</code>	No automatic unnamed captures	Only explicitly named captures are retained as captures

4.8 Flag behavior in one program

The final line shows `U` changing `<.*>` from a greedy match to the first tag-sized match.

Verified Kokum example · 17_flags.kokum

```
FUNCTION Main AS INTEGER
  LOCAL rows AS ARRAY

  ? "i=" + STR(REGEXMATCH("[a-z]+$", "kokum", "i"))
  rows = REGEXFINDALL("^ERROR.*$", "INFO\nERROR one\nERROR two", "m")
  ? "m=" + JSONENCODE(rows)
  ? "s=" + STR(REGEXMATCH("^start.*end$", "start\nend", "s"))
  ? "x=" + STR(REGEXMATCH("[A-Z]{2} - [0-9]{4} $" , "AB-1234", "x"))
  ? "u=" + STR(REGEXMATCH("^\\w+$", "Ω", "u"))
  ? "a=" + STR(REGEXMATCH("^\\w+$", "Ω", "a"))
  ? "U=" + REGEXFIND("<.*>", "<a><b>", "U")
  RETURN 0
ENDFUNC
```

Output

```
i=.T.
m=["ERROR one","ERROR two"]
s=.T.
x=.T.
u=.T.
a=.F.
```

```
U=<a>
```

5. Compiled REGEX resources

When the same pattern is applied repeatedly, compile it once. `REGEXCOMPILE()` returns a first-class native REGEX resource. `PATTERN` is accepted as a type-name alias.

Method	Arguments	Result
<code>IsMatch / Test</code>	subject	LOGICAL
<code>Find</code>	subject	STRING or NULL
<code>FindAll</code>	subject	ARRAY
<code>Replace</code>	subject, replacement	STRING
<code>Split</code>	subject	ARRAY
<code>Pattern</code>	none	Original pattern string
<code>Flags</code>	none	Effective canonical flag string
<code>IsClosed</code>	none	LOGICAL
<code>Close</code>	none	Closes the native resource

5.1 Compile, inspect, use, and close

The effective flag string includes the default Unicode flag, so compiling with `"i"` reports `ui`.

Verified Kokum example · 06_compiled.kokum

```
FUNCTION Main AS INTEGER
  LOCAL expression AS REGEX

  expression = REGEXCOMPILE("^a.+p$", "i")
  ? expression.IsMatch("APPLEP")
  ? expression.Pattern()
  ? expression.Flags()
  ? expression.IsClosed()
  expression.Close()
  ? expression.IsClosed()
  RETURN 0
ENDFUNC
```

Output

```
.T.
^a.+p$
ui
.F.
.T.
```

WARNING

Closed resources: Calling `Find`, `Replace`, or another matching method after `Close()` raises KRG1500. `IsClosed()` remains safe to call.

IMPORTANT

Immutable options: Do not supply a separate flags argument when passing a compiled REGEX to a direct function. Its compile options are already fixed.

6. RegEx with FILE subjects

The direct matching functions accept the ordinary **FILE** value returned by `OPEN ( )`. No RegEx-specific file type exists. Reading begins at the handle's current position, continues to EOF, advances the handle, and leaves it open for the caller to close.

FILE cursor behavior: direct RegEx versus streaming FIND

Direct REGEX... function



Natural FIND FROM FILE ... FIRST / LIMIT



Figure 4. Direct RegEx consumes the remaining FILE subject; streaming FIND can stop earlier.

6.1 Direct RegEx from the current FILE position

The first line is read normally. `REGEXFINDALL ( )` then sees only the remaining text and leaves the handle at EOF.

Fixture file

```
Header
APPLEP
BANANA
ALPHAP
```

Verified Kokum example · 07_file_direct.kokum

```
FUNCTION Main AS INTEGER
  LOCAL file AS FILE
  LOCAL heading
  LOCAL matches AS ARRAY

  file = OPEN("records.txt", "r")
  heading = READLINE(file)
  matches = REGEXFINDALL("^A.*P$", file, "m")

  ? heading
  ? JSONENCODE(matches)
  ? TYPEOF(READLINE(file))
  CLOSEFILE(file)
  RETURN 0
ENDFUNC
```

Output

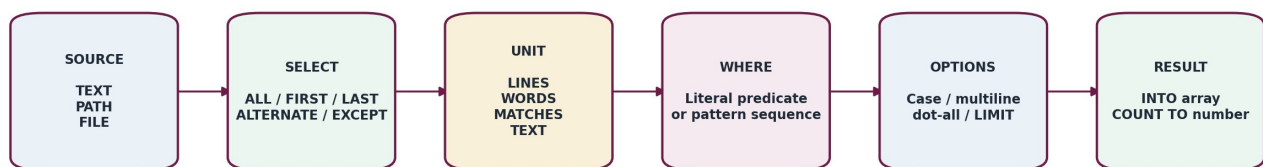
```
Header
["APPLEP", "ALPHAP"]
NULL
```

Rule	Behavior
Ownership	The caller owns and closes an existing FILE handle
Position	Direct RegEx reads from the current position to EOF
Mutation	REGEXREPLACE() returns a new string and never writes back automatically
Validation	Normal FILE mode, UTF-8, ownership, and error checks remain active
Path convenience	Natural Pattern statements can also receive a filename directly and manage opening/closing internally

7. Natural Pattern fundamentals

Natural Pattern statements describe search and transformation with contextual Kokum keywords. Words such as **find**, **replace**, **split**, **validate**, **iterate**, and **capture** remain legal variable names outside the corresponding statement forms.

How a Natural FIND statement is evaluated



LINES and WORDS can stream. LIMIT may stop work early. WITH DETAILS changes each result into a match map.

Figure 5. Natural FIND separates source, selection, unit, condition, options, and destination.

7.1 Source forms

Form	Meaning
FROM TEXT expression	Use an in-memory STRING expression
FROM "records.txt"	Open a path internally and close it on success or error
FROM FILE fileHandle	Use an already-open ordinary FILE at its current position

7.2 Selectors

Selector	Meaning
ALL	Every source unit or match
FIRST n	The first n candidates
LAST n	The last n candidates
ALTERNATE	Candidates 1, 3, 5, ...
ALTERNATE STARTING WITH n	Every other candidate beginning at one-based index n
EXCEPT FIRST [n]	Skip the first n; omitted n means 1
EXCEPT LAST [n]	Skip the last n; omitted n means 1

7.3 FIND units

Unit	Candidate value
LINE	Each physical line without its line terminator
WORDS	Unicode-aware word units
MATCHES	Every non-overlapping matching substring
TEXT	The complete remaining source as one candidate

7.4 Literal predicates

Predicate	Meaning
STARTS WITH value	Candidate begins with literal value
STARTING FROM value	Accepted compatibility alias
ENDS WITH value	Candidate ends with literal value
ENDING WITH value	Accepted compatibility alias
CONTAINS value	Candidate contains literal value
EQUALS value	Candidate equals literal value
IS BLANK / IS NOT BLANK	Empty or non-empty candidate; no implicit trimming
MATCHES REGEX expression	Use a raw STRING pattern or compiled REGEX

7.5 Pattern atoms

Atom	Meaning
LETTER	Unicode letter
DIGIT	Digit
WHITESPACE	Whitespace
WORD CHARACTER	Word character
ANY CHARACTER	Any character
NEWLINE	Unicode newline sequence in MATCHES/TEXT contexts
TAB	Tab character
ONE OF "..."	One character from a literal set
NONE OF "..."	One character outside a literal set
CHARACTER BETWEEN "A" AND "Z"	One character in an inclusive range
START OF TEXT / END OF TEXT	Absolute subject boundaries
START OF LINE / END OF LINE	Line boundaries
WORD BOUNDARY	Boundary between word and non-word text
"literal"	Literal sequence atom; metacharacters are escaped automatically

7.6 Quantifiers and sequencing

Natural form	Count
OPTIONAL atom	Zero or one
ZERO OR MORE atom(s)	Zero or more
ONE OR MORE atom(s)	One or more
EXACTLY n atom(s)	Exactly n
AT LEAST n atom(s)	n or more
AT MOST n atom(s)	Zero through n
BETWEEN minimum AND maximum atom(s)	Inclusive range
... FOLLOWED BY ...	Concatenate pattern atoms in order

Natural Pattern sequence
EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY BETWEEN 2 AND 4 DIGITS

7.7 Boolean conditions and precedence

Boolean condition precedence is: parentheses, **NOT**, **AND**, then **OR**. Boolean trees are supported for **LINES**, **WORDS**, **TEXT**, and **VALIDATE**. Substring-producing **MATCHES**, **REPLACE**, **SPLIT**, and **ITERATE** require one unambiguous pattern sequence or raw RegEx leaf and reject Boolean trees with **KRG1305**.

WARNING

NEWLINE with LINES: A **LINES** unit no longer contains its terminator. Use **IS BLANK** to locate an empty line. Use **NEWLINE** in **MATCHES/TEXT** or **SPLIT** when the terminator itself must be matched.

8. FIND statements

Canonical FIND syntax
<pre> FIND FROM <source> <selection> <unit> WHERE <condition> [CASE SENSITIVE IGNORE CASE] [MULTILINE] [DOT MATCHES NEWLINE] [LIMIT <positive-integer-expression>] [WITH DETAILS] (INTO <array-target> COUNT TO <numeric-target>) </pre>

8.1 Search lines in memory

The input is divided into physical lines and the line terminators are removed before the predicate is evaluated.

Verified Kokum example · 08_natural_find_lines.kokum
<pre> FUNCTION Main AS INTEGER LOCAL text AS STRING LOCAL rows AS ARRAY text = "A1\nB2\nA3\nB4" FIND FROM TEXT text ALL LINES WHERE STARTS WITH "A" INTO rows ? JSONENCODE(rows) </pre>

```
RETURN 0
ENDFUNC
```

Output

```
["A1", "A3"]
```

8.2 Search words and matches; count without retaining

COUNT TO is preferred when only the number of matches is required because it does not retain an output array.

Verified Kokum example · 09_natural_units_count.kokum

```
FUNCTION Main AS INTEGER
  LOCAL words AS ARRAY
  LOCAL codes AS ARRAY
  LOCAL count AS INTEGER

  FIND FROM TEXT "Apple Apricot Banana Avocado" ALL WORDS
    WHERE STARTS WITH "A"
    INTO words

  FIND FROM TEXT "AA12 BB340 cc7" ALL MATCHES
    WHERE EXACTLY 2 LETTERS FOLLOWED BY ONE OR MORE DIGITS
    INTO codes

  FIND FROM TEXT "A1\nB2\nA3" ALL LINES
    WHERE CONTAINS "A"
    COUNT TO count

  ? JSONENCODE(words)
  ? JSONENCODE(codes)
  ? count
  RETURN 0
ENDFUNC
```

Output

```
["Apple", "Apricot", "Avocado"]
["AA12", "BB340", "cc7"]
2
```

8.3 Selector order matters

For line and word searches, the selector chooses candidates before the **WHERE** predicate. Therefore **FIRST 3 LINES** can return fewer than three rows if some of those first three lines do not satisfy the predicate.

8.4 FIRST and ALTERNATE selectors

The first search selects lines 1–3 and then keeps the A-lines. The second selects lines 2 and 4 before checking the B-prefix.

Verified Kokum example · 20_natural_selectors.kokum

```
FUNCTION Main AS INTEGER
  LOCAL rows AS ARRAY

  FIND FROM TEXT "A1\nB2\nA3\nB4\nA5" FIRST 3 LINES
    WHERE STARTS WITH "A"
    INTO rows
  ? "first=" + JSONENCODE(rows)

  FIND FROM TEXT "A1\nB2\nA3\nB4\nA5" ALTERNATE LINES STARTING WITH 2
    WHERE STARTS WITH "B"
    INTO rows
  ? "alternate=" + JSONENCODE(rows)
  RETURN 0
ENDFUNC
```

Output

```
first=["A1", "A3"]
alternate=["B2", "B4"]
```

Option	FIND meaning
IGNORE CASE	Case-insensitive condition
CASE SENSITIVE	Explicit case-sensitive condition
MULTILINE	Line semantics for raw anchors where applicable
DOT MATCHES NEWLINE	Raw dot and generated any-character forms can cross newline
LIMIT n	Return at most n successful results and permit early termination
WITH DETAILS	Return a MAP for each match instead of a STRING

8.5 Streaming an existing FILE

LINES and **WORDS** use incremental **FILE** processing. When **FIRST** or **LIMIT** satisfies the request early, the file cursor may remain before EOF, allowing ordinary reads to continue.

8.6 Continue reading after a streaming FIND

After the heading, **FIRST 2 LINES** consumes APPLEP and BANANA. The next ordinary **READLINE()** therefore returns ALPHAP.

Fixture file

```
Header
APPLEP
BANANA
ALPHAP
```

Verified Kokum example · 15_file_stream.kokum

```
FUNCTION Main AS INTEGER
  LOCAL file AS FILE
  LOCAL heading
  LOCAL rows AS ARRAY
  LOCAL nextLine

  file = OPEN("records.txt", "r")
  heading = READLINE(file)
  FIND FROM FILE file FIRST 2 LINES
    WHERE IS NOT BLANK
    INTO rows
  nextLine = READLINE(file)

  ? heading
  ? JSONENCODE(rows)
  ? nextLine
  CLOSEFILE(file)
  RETURN 0
ENDFUNC
```

Output

```
Header
["APPLEP", "BANANA"]
ALPHAP
```

9. Captures and detailed results

Natural **CAPTURE** declarations retain a quantified atom without requiring raw RegEx parentheses. Add **AS name** on the same logical source line to create a named capture.

Natural capture declaration

```
CAPTURE EXACTLY 2 LETTERS AS code
FOLLOWED BY "-"
FOLLOWED BY CAPTURE EXACTLY 4 DIGITS AS number
```

Natural captures and the detailed match map

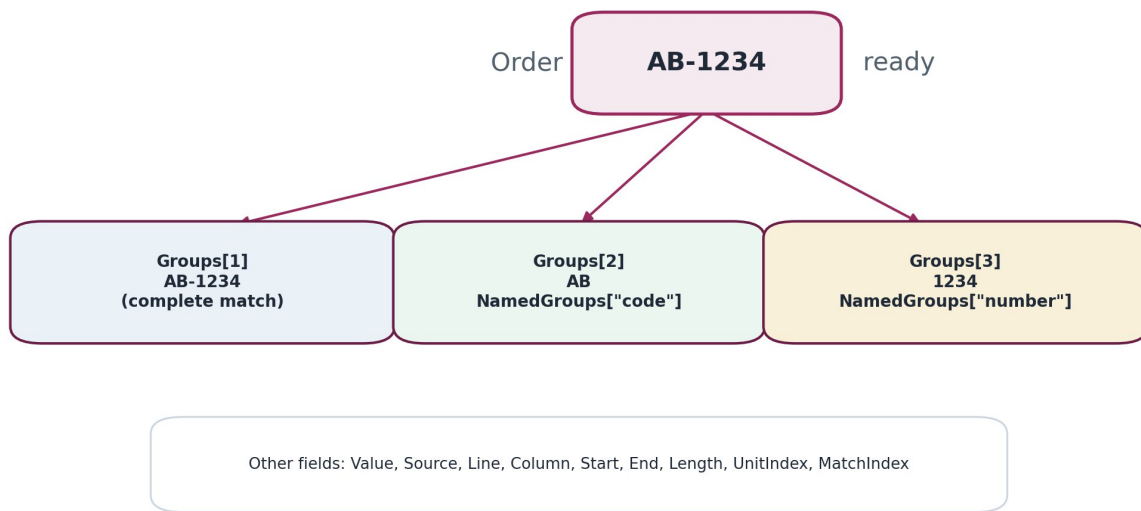


Figure 6. Groups are stored in a one-based Kokum ARRAY with the complete match first.

IMPORTANT

Group indexing: `Groups[1]` is the complete match. The first declared capture is `Groups[2]`, the second is `Groups[3]`, and so on. `NamedGroups["name"]` addresses a named capture directly.

9.1 Read natural capture values

WITH DETAILS changes each array item into a MAP. This example reads the complete match, the first captured group, and a named group.

Verified Kokum example · 10_details_capture.kokum

```
FUNCTION Main AS INTEGER
  LOCAL details AS ARRAY

  FIND FROM TEXT "Order AB-1234" ALL MATCHES
    WHERE CAPTURE EXACTLY 2 LETTERS AS code FOLLOWED BY "-" FOLLOWED BY CAPTURE EXACTLY 4
DIGITS AS number
    WITH DETAILS
    INTO details

  ? details[1].Value
  ? details[1].Groups[2]
  ? details[1].NamedGroups["number"]
  RETURN 0
ENDFUNC
```

Output

```
AB-1234
AB
1234
```

9.2 Stable detail fields

Field	Meaning
Value	Complete matched string
Source	Source descriptor associated with the match
Line / Column	One-based position for human-facing reporting
Start / End / Length	UTF-8 byte measurements; Length is the matched byte length
UnitIndex	One-based source-unit position
MatchIndex	One-based match sequence position
Groups	Complete match followed by numbered captures
NamedGroups	MAP from capture name to captured value

9.3 Raw named groups with FIND

```
Compiled RegEx inside a Natural FIND
LOCAL rx AS REGEX
LOCAL details AS ARRAY

rx = REGEXCOMPILE("(?<code>[A-Z]{2})-(?<number>[0-9]{4})")
FIND FROM TEXT "Order AB-1234" ALL MATCHES
  WHERE MATCHES REGEX rx
  WITH DETAILS
  INTO details
rx.Close()
```

This form is useful when an advanced raw pattern already exists but Natural **FIND** selectors, FILE semantics, positions, and detail maps are still desirable.

10. Natural REPLACE

```
Natural REPLACE syntax
REPLACE FROM <source>
  [<selection> MATCH | MATCHES]
  WHERE <single-pattern-condition>
  WITH <string-expression>
  [CASE SENSITIVE | IGNORE CASE]
  [MULTILINE]
  [DOT MATCHES NEWLINE]
  [LIMIT <positive-integer-expression>]
  INTO <assignment-target>
```

A missing selector means **ALL MATCHES**. Selection is computed over the non-overlapping match sequence. **LIMIT** then restricts how many selected matches are replaced. The original string or FILE contents are never modified implicitly.

10.1 Reformat codes with named and numbered captures

The result is assigned to a new string. The original source expression remains unchanged.

```
Verified Kokum example · 11_natural_replace.kokum
FUNCTION Main AS INTEGER
  LOCAL output AS STRING
```

```

REPLACE FROM TEXT "AA12 BB340 cc7" ALL MATCHES
  WHERE CAPTURE EXACTLY 2 LETTERS AS code FOLLOWED BY CAPTURE ONE OR MORE DIGITS AS number
  WITH "${code}:%2"
  INTO output

```

```

? output
RETURN 0
ENDFUNC

```

Output

```
AA:12 BB:340 cc:7
```

Reference	Natural REPLACE meaning
\$0 or \$&	Complete selected match
\$1, \$2, ...	Numbered natural or raw capture
\${name} or \$<name>	Named capture
\$\$	Literal dollar sign

WARNING

Strict capture references: A missing numbered or named reference raises **KRG1306** in Natural REPLACE. This prevents accidental data loss from a misspelled capture name.

10.2 Replacement selection and LIMIT

Statement fragment	Effect
FIRST 2 MATCHES ... WITH "\$0"	Replace the first two matches
LAST 2 MATCHES	Replace the final two matches
ALTERNATE MATCHES STARTING WITH 2	Replace matches 2, 4, 6, ...
EXCEPT FIRST 2 MATCHES	Leave the first two matches unchanged
ALL MATCHES ... LIMIT 2	Replace at most two selected matches

10.3 Zero-width replacements

Boundary atoms may match without consuming text. For example, `REPLACE FROM TEXT "abc" WHERE START OF TEXT WITH ">" INTO output` produces `>abc`. Kokum advances global matching safely after zero-length matches.

11. Natural SPLIT

Natural SPLIT syntax

```

SPLIT FROM <source>
  WHERE <single-pattern-condition>
  [KEEP DELIMITERS]
  [CASE SENSITIVE | IGNORE CASE]
  [MULTILINE]
  [DOT MATCHES NEWLINE]
  [LIMIT <positive-integer-expression>]
  INTO <assignment-target>

```

LIMIT is the maximum number of delimiters consumed, not the number of returned fields. Empty leading, interior, and trailing fields are retained. When no delimiter matches, the result is a one-element array containing the original source.

11.1 Keep delimiters in the result

KEEP DELIMITERS inserts each matched delimiter between the surrounding fields.

Verified Kokum example · 12_natural_split.kokum

```
FUNCTION Main AS INTEGER
  LOCAL parts AS ARRAY

  SPLIT FROM TEXT "one,two;three"
    WHERE ONE OF ",";"
    KEEP DELIMITERS
    INTO parts

  ? JSONENCODE(parts)
  RETURN 0
ENDFUNC
```

Output

```
["one", ",", "two", ";", "three"]
```

11.2 Delimiter limits and empty fields

After two delimiter matches, the unprocessed tail remains one field. Adjacent and trailing delimiters produce empty strings.

Verified Kokum example · 21_split_edges.kokum

```
FUNCTION Main AS INTEGER
  LOCAL parts AS ARRAY

  SPLIT FROM TEXT "a,b,c,d" WHERE "," LIMIT 2 INTO parts
  ? JSONENCODE(parts)

  SPLIT FROM TEXT "a,,b," WHERE "," INTO parts
  ? JSONENCODE(parts)
  RETURN 0
ENDFUNC
```

Output

```
["a", "b", "c, d"]
["a", "", "b", ""]
```

11.3 Split physical lines

Path-based line split

```
LOCAL parts AS ARRAY
SPLIT FROM "records.txt"
  WHERE NEWLINE
  LIMIT 2
  INTO parts
```

NOTE

LINES versus NEWLINE: Use **NEWLINE** when splitting the complete text. Do not use a bare **NEWLINE** predicate after **FIND ... LINES**, because line terminators have already been removed there.

12. VALIDATE

VALIDATE syntax

```
VALIDATE FROM <source>
  WHERE <condition>
  [CASE SENSITIVE | IGNORE CASE]
  [MULTILINE]
  [DOT MATCHES NEWLINE]
  INTO <assignment-target>
```

VALIDATE returns a LOGICAL. Natural pattern sequences and MATCHES REGEX leaves are implicitly anchored to the complete remaining source. Literal predicates such as STARTS WITH, CONTAINS, and EQUALS keep their direct meanings.

12.1 Validate a structured code

No explicit start or end atoms are needed; the pattern sequence must cover the entire subject.

Verified Kokum example · 13_validate.kokum

```
FUNCTION Main AS INTEGER
  LOCAL valid AS LOGICAL

  VALIDATE FROM TEXT "AB-1234"
    WHERE EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY EXACTLY 4 DIGITS
    INTO valid

  ? valid
  RETURN 0
ENDFUNC
```

Output

.T.

12.2 Boolean validation and dot-all matching

VALIDATE permits Boolean conditions. DOT MATCHES NEWLINE allows a raw dot to span the embedded line break.

Verified Kokum example · 22_validate_boolean.kokum

```
FUNCTION Main AS INTEGER
  LOCAL valid AS LOGICAL

  VALIDATE FROM TEXT "APPLEP"
    WHERE STARTS WITH "A" AND ENDS WITH "P"
    INTO valid
  ? valid

  VALIDATE FROM TEXT "abc\ndef"
    WHERE MATCHES REGEX "abc.def"
    DOT MATCHES NEWLINE
    INTO valid
  ? valid
  RETURN 0
ENDFUNC
```

Output

.T.
.T.

Condition	Validation behavior
EXACTLY 2 LETTERS FOLLOWED BY EXACTLY 2 DIGITS	Whole-source pattern sequence

Condition	Validation behavior
MATCHES REGEX "[A-Z]{2}[0-9]{2}"	Whole-source raw RegEx leaf
STARTS WITH "A" AND ENDS WITH "P"	Boolean combination of direct predicates
IS BLANK	True only for the empty string; spaces and tabs are not trimmed

TIP

Prefer VALIDATE for forms: For text-box or configuration validation, **VALIDATE** communicates whole-value intent more clearly than remembering anchors around every raw pattern.

13. ITERATE

ITERATE syntax

```

ITERATE FROM <source>
  [<selection> MATCH | MATCHES]
  WHERE <single-pattern-condition>
  [CASE SENSITIVE | IGNORE CASE]
  [MULTILINE]
  [DOT MATCHES NEWLINE]
  [LIMIT <positive-integer-expression>]
  AS <match-variable> [WITH INDEX <index-variable>]
  <ordinary Kokum statements>
ENDITERATE [<match-variable>]

```

The match variable is the same detail MAP produced by **FIND ... WITH DETAILS**. The optional index is one-based. The body is ordinary Kokum code and may use normal control flow, functions, arrays, maps, and application state.

13.1 Process each match with an index

The iterator provides the complete value and named captures without first materializing an array in user code.

Verified Kokum example · 14_iterate.kokum

```

FUNCTION Main AS INTEGER
  LOCAL item AS MAP
  LOCAL position AS INTEGER

  ITERATE FROM TEXT "AA12 BB3" ALL MATCHES
    WHERE CAPTURE EXACTLY 2 LETTERS AS code FOLLOWED BY CAPTURE ONE OR MORE DIGITS AS number
    AS item WITH INDEX position
    ? STR(position) + ":" + item.Value + ":" + item.NamedGroups["code"]
  ENDITERATE item

  RETURN 0
ENDFUNC

```

Output

```

1:AA12:AA
2:BB3:BB

```

Rule	Meaning
ENDITERATE	Ends the iteration block
END ITERATE	Accepted two-word terminator
ENDITERATE item	Optional variable name must match the iterator variable
AS item WITH INDEX position	Creates a detail MAP and one-based numeric index

Rule	Meaning
LIMIT n	Yield at most n selected matches

IMPORTANT

FILE iteration: Natural **ITERATE**, **REPLACE**, **SPLIT**, and **VALIDATE** are whole-subject operations. With an existing **FILE** they consume from the current position to EOF and leave the handle open at EOF.

14. IDE assistance and Pattern Inspector

The Kokum IDE recognizes **FIND**, **REPLACE**, **SPLIT**, **VALIDATE**, and complete **ITERATE** blocks contextually. Natural Pattern words remain ordinary identifiers elsewhere.

Action	Shortcut / control	Purpose
Completion	Ctrl+Space / Command+Space	Suggest source forms, selectors, units, conditions, options, and destinations
Format statement	Ctrl+Shift+F / Command+Shift+F	Normalize layout while preserving quoted strings and escapes
Pattern Inspector	Ctrl+Alt+R	Inspect the current statement and a sample subject
Toolbar	Pattern	Open the Natural Pattern Inspector
Compiler Check	Check / Build / Run / Deploy	Authoritative parser and semantic diagnostics

The Pattern Inspector shows the normalized statement, operation, source, selector, unit, flags, destination, generated-RegEx preview where available, structural diagnostics, and sample results. Immediate editor feedback is helpful, but **kokumc check** remains authoritative.

14.1 Compiler-assisted command-line bridge

Developer utility

```
python3 tools/kokum_pattern_inspector_bridge.py --kokumc ./kokumc --statement 'FIND FROM TEXT
"APPLEP" ALL LINES WHERE STARTS WITH "A" INTO result' --sample '$APPLEP
BANANA'
```

NOTE

Native tooling: The bridge delegates parsing and sample execution to **kokumc**. It does not require Node.js or a second server-side JavaScript grammar implementation.

15. Diagnostics and troubleshooting

Direct RegEx runtime errors and Natural Pattern compiler/runtime errors use the **KRG** diagnostic family. Run **kokumc check** before execution to catch structural errors early.

15.1 Catch an invalid raw pattern

RegEx compile failures are normal Kokum errors and may be handled with **TRY**, **CATCH**, and **ENDTRY**.

Verified Kokum example · 18_error_handling.kokum

```
FUNCTION Main AS INTEGER
  TRY
    REGEXCOMPILE("([A-Z])")
  CATCH error
    ? STR(error)
  ENDTRY
  RETURN 0
```

ENDFUNC

Output

KRG1100: invalid regular expression at byte 7: missing closing parenthesis

Code	Typical cause	Correction
KRG1001	Wrong arity, malformed operation, or missing ENDITERATE	Check the canonical syntax and closing block
KRG1002	Wrong source, pattern, or replacement type	Use STRING , FILE , REGEX , or the required destination type
KRG1003 / KRG1005	Invalid, duplicate, or conflicting flags	Remove duplicates; never combine u and a
KRG1006	Duplicate LIMIT or KEEP DELIMITERS	Keep each option once
KRG1009	ENDITERATE variable mismatch	Use the same iterator name or omit the trailing name
KRG1100	Raw/generated pattern cannot compile	Inspect the reported byte position and grouping/escaping
KRG1104	NEWLINE used after division into LINES	Use IS BLANK or search MATCHES/TEXT
KRG1203	Invalid natural capture declaration/name	Use a valid identifier on the same logical line
KRG1302	LIMIT is zero or negative	Use a positive integer
KRG1305	Boolean tree where one substring pattern is required	Use one sequence/raw RegEx, or move the Boolean logic to VALIDATE
KRG1306	Missing replacement group reference	Correct \$n , \${name} , or <name>
KRG1500	Compiled REGEX was closed	Recompile or avoid using the resource after Close
KRG1501	Missing destination/iterator or runtime resource issue	Add INTO , COUNT TO , AS item, or correct resource use
KRG170x	Configured size, match, depth, or output limit exceeded	Use streaming, LIMIT , a simpler pattern, or smaller input

15.2 Common mistakes

Mistake	Symptom	Fix
REGEXMATCH ("[0-9]+", "A12") for strict validation	Returns true because a substring matches	Anchor it or use VALIDATE
"\d+" written with one source backslash	String/parser confusion or wrong runtime pattern	Write "\\d+" in Kokum source
User input inserted directly into a raw pattern	Dots, brackets, or plus signs gain RegEx meaning	Use REGEXESCAPE
FIND ... LINES WHERE NEWLINE	KRG1104	Use IS BLANK or MATCHES/TEXT
Boolean OR inside REPLACE/SPLIT/ITERATE	KRG1305	Use a single pattern such as ONE OF or a raw alternation
Flags supplied with an already compiled REGEX	Immutable-option diagnostic	Compile with the required flags once
Expecting a FILE replacement to modify disk	Source file remains unchanged	Write the returned string explicitly

16. Performance, safety, and portability

The RegEx implementation applies bounded defaults to prevent hostile or accidental patterns from exhausting memory or native recursion. These are implementation safety limits, not targets for routine applications.

Limit	Default
Pattern size	64 KiB
Whole subject size	64 MiB
Returned/materialized matches	100,000
PCRE2 match operations	10,000,000
PCRE2 depth	1,000
Natural parenthesized nesting	1,024
Natural Boolean/NOT operators	2,048

16.1 Choose streaming where possible

- Use `FIND ... LINES` or `FIND ... WORDS` for large text files.
- Use `COUNT TO` instead of `INTO` when values are not required.
- Use `LIMIT` to permit early completion.
- Remember that direct RegEx, `MATCHES/TEXT` whole-subject work, and natural transformation operations remain bounded by the subject-size limit.

16.2 Compile repeated raw patterns

Use `REGEXCOMPILE()` when the same raw pattern is used in a loop, request handler, form validation routine, or batch job. Keep it open while needed and call `Close()` when its lifetime is complete.

16.3 Prefer clear patterns

- Anchor validation patterns so accidental substrings are not accepted.
- Use explicit character classes instead of an overly broad dot when possible.
- Avoid nested ambiguous repetition such as repeated `.*` structures.
- Use Natural Pattern literals when manual escaping obscures the intent.

16.4 Engine parity

Direct RegEx and Natural Pattern operations lower through the same implementation for source interpretation, canonical IR, bytecode execution, external KokumVM, application bundles, and standalone executables. The public built-in IDs and artifact formats remain stable across the documented phase.

NOTE

Native dependency boundary: PCRE2 is isolated behind Kokum's RegEx API. Application code uses Kokum functions and types rather than PCRE2 headers. Node.js is not part of the RegEx runtime.

17. Practical recipes

The following examples illustrate common application tasks. Validation patterns that check a shape do not automatically prove semantic validity—for example, a date-shaped value still requires calendar validation if month lengths matter.

17.1 Username validation, code extraction, and error-log filtering

This single program demonstrates a compact raw validation pattern and two readable Natural FIND operations.

Verified Kokum example · 16_recipes.kokum

```
FUNCTION Main AS INTEGER
  LOCAL ok AS LOGICAL
  LOCAL codes AS ARRAY
  LOCAL errors AS ARRAY
```

```
ok = REGEXMATCH("[A-Za-z][A-Za-z0-9_]{2,15}$", "Kokum_29")
? "username=" + STR(ok)

FIND FROM TEXT "Invoices AB-1234 and XY-9876" ALL MATCHES
  WHERE EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY EXACTLY 4 DIGITS
  INTO codes
? "codes=" + JSONENCODE(codes)

FIND FROM TEXT "INFO started\nERROR disk full\nWARN slow\nERROR stopped" ALL LINES
  WHERE STARTS WITH "ERROR"
  INTO errors
? "errors=" + JSONENCODE(errors)
RETURN 0
ENDFUNC
```

Output

```
username=.T.
codes=["AB-1234","XY-9876"]
errors=["ERROR disk full","ERROR stopped"]
```

17.2 Recipe table

Task	Kokum pattern or statement	Note
Signed integer	<code>^[+-]?[0-9]+\$</code>	Use with REGEXMATCH or REGEXCOMPILE
Simple decimal shape	<code>^[+-]?(?:[0-9]+(?:\.[0-9]*)?) \.[0-9]+\$</code>	In Kokum source, each RegEx backslash is doubled
ISO date shape	<code>^[0-9]{4}-[0-9]{2}-[0-9]{2}\$</code>	Checks shape only
Username	<code>^[A-Za-z][A-Za-z0-9_]{2,15}\$</code>	3–16 characters; begins with a letter
Case-insensitive CSV extension	<code>REGEXMATCH("\\.csv\$", name, "i")</code>	Tests filename suffix
Normalize whitespace	<code>REGEXREPLACE("\\s+", text, " ")</code>	Returns a new string
Extract product codes	<code>FIND ... ALL MATCHES WHERE EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY EXACTLY 4 DIGITS</code>	Readable and Unicode-letter aware
Filter error lines	<code>FIND ... ALL LINES WHERE STARTS WITH "ERROR"</code>	Streams FILE input in line mode
Split comma or semicolon	<code>SPLIT ... WHERE ONE OF ",;"</code>	Add KEEP DELIMITERS when separators are significant
Validate whole code	<code>VALIDATE ... WHERE EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY EXACTLY 4 DIGITS</code>	Implicit whole-source anchoring

17.3 Selecting direct or Natural form for the same task

Intent	Direct form	Natural form
Find codes	<code>REGEXFINDALL("[A-Z]{2}[0-9]+", text)</code>	<code>FIND FROM TEXT text ALL MATCHES WHERE EXACTLY 2 LETTERS FOLLOWED BY ONE OR MORE DIGITS INTO rows</code>
Validate code	<code>REGEXMATCH("^([A-Z]{2}-[0-9]{4})\$", text)</code>	<code>VALIDATE FROM TEXT text WHERE EXACTLY 2 LETTERS FOLLOWED BY "-" FOLLOWED BY EXACTLY 4 DIGITS INTO valid</code>
Replace codes	<code>REGEXREPLACE("([A-Z]{2})([0-9]+)", text, "\$1:\$2")</code>	<code>REPLACE FROM TEXT text ALL MATCHES WHERE CAPTURE EXACTLY 2 LETTERS FOLLOWED BY CAPTURE ONE OR MORE DIGITS WITH "\$1:\$2" INTO output</code>

TIP

Team conventions: A project can use both interfaces. A practical convention is to prefer Natural Pattern for business-readable rules and raw RegEx for compact standard patterns, advanced grouping, or imported expressions.

Appendix A. Direct API quick reference

API	Arity	Return	No-match behavior
REGEXCOMPILE	1–2	REGEX	Compile error raises KRG1100
REGEXMATCH	2–3	LOGICAL	.F.
REGEXFIND	2–3	STRING or NULL	NULL
REGEXFINDALL	2–3	ARRAY	Empty array
REGEXREPLACE	3–4	STRING	Original string
REGEXSPLIT	2–3	ARRAY	One field containing original string
REGEXESCAPE	1	STRING	Not applicable

Appendix B. Natural Pattern operation matrix

Feature	FIND	REPLACE	SPLIT	VALIDATE	ITERATE
PATH / TEXT / FILE sources	Yes	Yes	Yes	Yes	Yes
Selectors	Yes	Match selectors	No	No	Match selectors
LINES / WORDS / MATCHES / TEXT units	Yes	Matches	Delimiters	Whole source	Matches
Boolean WHERE	LINES/WORDS/TEXT	No	No	Yes	No
CAPTURE	With details	Yes	Pattern allowed	Pattern allowed	Yes
LIMIT	Results	Replacements	Delimiters	No	Yielded matches
WITH DETAILS	Yes	No	No	No	Always detail MAP
KEEP DELIMITERS	No	No	Yes	No	No
Destination	INTO / COUNT TO	INTO	INTO	INTO	AS variable

Appendix C. Natural Pattern vocabulary

Category	Vocabulary
Sources	"path", FILE expression, TEXT expression
Selectors	ALL, FIRST n, LAST n, ALTERNATE [STARTING WITH n], EXCEPT FIRST [n], EXCEPT LAST [n]
Units	LINES, WORDS, MATCHES, TEXT
Predicates	STARTS WITH, STARTING FROM, ENDS WITH, ENDING WITH, CONTAINS, EQUALS, IS BLANK, IS NOT BLANK, MATCHES REGEX
Atoms	LETTER, DIGIT, WHITESPACE, WORD CHARACTER, ANY CHARACTER, NEWLINE, TAB, ONE OF, NONE OF, CHARACTER BETWEEN, text/line anchors, WORD BOUNDARY, quoted literals
Quantifiers	OPTIONAL, ZERO OR MORE, ONE OR MORE, EXACTLY n, AT LEAST n, AT MOST n, BETWEEN m AND n
Composition	FOLLOWED BY, CAPTURE ... [AS name], parentheses, NOT, AND, OR
Options	CASE SENSITIVE, IGNORE CASE, MULTILINE, DOT MATCHES NEWLINE, LIMIT, WITH DETAILS, KEEP DELIMITERS

Appendix D. Detailed result map reference

Field	Type	Index basis	Use
Value	STRING	—	Complete match
Source	STRING	—	Source label/path context
Line	INTEGER	1-based	Human-readable line
Column	INTEGER	1-based	Human-readable column
Start	INTEGER	0-based bytes	Start byte offset
End	INTEGER	0-based bytes	End byte offset
Length	INTEGER	bytes	Matched UTF-8 byte length
UnitIndex	INTEGER	1-based	Source unit number
MatchIndex	INTEGER	1-based	Match number
Groups	ARRAY	Kokum 1-based	Groups[1] is complete match
NamedGroups	MAP	name key	Named capture lookup

Appendix E. Glossary

Term	Meaning
Anchor	A zero-width rule that fixes a match to a boundary such as start or end
Atom	One smallest Natural Pattern component, such as LETTER, DIGIT, or a literal
Capture	A retained portion of a match, addressed by number or name
Delimiter	A matched separator consumed by a split operation
Flags	Compile options such as Unicode, case-insensitive, multiline, or dot-all
Match	A successful substring produced by a pattern
Natural Pattern	Kokum's typed readable pattern syntax compiled to the native RegEx runtime
Pattern	The rule used to examine a subject
RegEx	Kokum's direct regular-expression API and first-class compiled resource type
Subject	The STRING or FILE text being examined
Unit	A FIND candidate: line, word, match, or complete text

Verification statement

The complete examples used in this manual were generated as standalone Kokum source files. Each was processed with `kokumc check`, executed with `kokumc run`, compiled to portable bytecode, verified with `kokumvm verify`, and executed with the external `kokumvm`. Source and VM output matched for all 22 programs.

Verified area	Examples
Direct API	Match, find, find-all, replace, split, escape, Unicode, flags, and named replacement
Resources	Compile, inspect effective flags, test, close, and closed-state query
FILE	Direct RegEx to EOF and streaming Natural FIND with continued reading
Natural FIND	LINES, WORDS, MATCHES, selectors, COUNT TO, and details

Verified area	Examples
Phase 4.16.7 operations	CAPTURE, REPLACE, SPLIT, VALIDATE, and ITERATE
Diagnostics	Catchable invalid-pattern diagnostic

NOTE

Source baseline: The verification used the latest Phase 4.16.7 Node.js-free source tree containing the Windows PCRE2 auto-provision correction. That build-system correction does not alter the language syntax documented here.

END OF MANUAL