

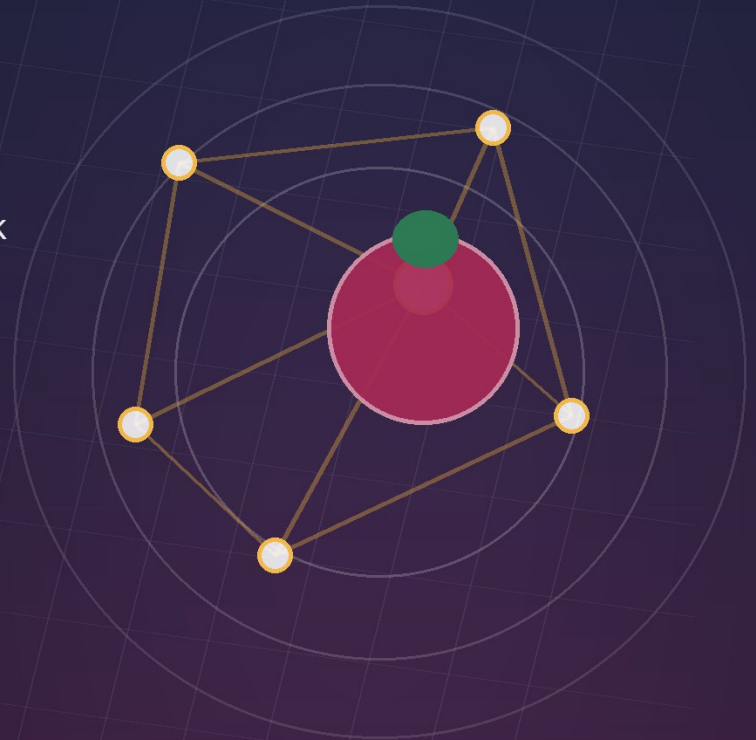
TIGERDB SOLUTIONS PRIVATE LIMITED

KOKUM

Core Language Programming

A professional manual and practical book

Kokum Basics Series 1.2



Syntax • Types • Collections • Functions • Modules
Databases • WebSockets • Concurrency • Microservices

Core language only — visual GUI designing is intentionally outside scope.

Verified examples: source check • interpreter • bytecode • KokumVM • bundles • services

DOCUMENT PROFILE

About this book

Kokum Core Language Programming is a standalone manual and practical book for developers who want to write console programs, reusable modules, data-processing applications, concurrent workloads, and backend services in Kokum. It documents version 0.29.0 through Phase 4.16.7.

SCOPE What is deliberately included and excluded

The book covers the executable core language and nonvisual application facilities. Visual form designing, GUI widgets, form events, and designer workflows are intentionally excluded. Regular Expressions and Natural Patterns are documented in the separate Kokum pattern-matching manual.

Profile item	Value
Language edition	Kokum 0.29.0, Phase 4.16.7
Source baseline	Node.js-free final working tree with Windows PCRE2 auto-provisioning
Primary platforms	Linux toolchain; portable KokumVM sources for Linux, macOS, and Windows
Document date	September 2026
Audience	Developers, trainers, testers, architects, and technical students
Examples	Small, complete programs plus verified multi-file projects and services

How the examples were verified

Every complete standalone listing was checked semantically, executed through the source interpreter, compiled to portable bytecode, and executed through the external KokumVM. The interpreter and VM outputs were compared. Project examples were also built as modules, bundles, and standalone executables where applicable.

Verification layer	Evidence used in this edition
Standalone programs	<code>kokumc check</code> , <code>kokumc run</code> , <code>kokumc compile</code> , and <code>kokumvm</code>
Modules and projects	project check, build, bundle, bundle execution, standalone build and execution
Remote FlatDB	local server, authenticated connection, DDL/DML/query execution in source and bytecode modes
WebSocket client	source/bytecode compilation plus the official local <code>ws://</code> and <code>wss://</code> loopback integration suite
HTTP microservice	bundle build, service startup, health endpoint, GET/POST/route-parameter requests, graceful stop

READING NOTE About output shown in the book

Observed output is copied from the verification runs. Values such as dates, request identifiers, task timing states, and garbage-collection counts can vary by run; examples focus on the stable behavior.

Suggested reading paths

Reader	Recommended path
New to Kokum	Chapters 1-13, then 16-18; return to objects and concurrency when needed.
Experienced programmer	Chapters 1-7 for language differences, then modules, databases, concurrency, and services.
Backend developer	Chapters 16-18, 20-26, with the collection and error-handling chapters as reference.
Trainer or reviewer	Read in order and use the chapter practice prompts as lab assignments.

CONTENTS

Table of contents

The page numbers are generated from the final rendered edition. Major headings use Word heading styles for navigation and accessibility.

PART I	LANGUAGE FOUNDATIONS	5
1	First Program and Execution Model	6
2	Source Structure, Comments, and Lexical Rules	8
3	Kokum Files, Projects, and Build Artifacts	10
4	Types and Literals	12
5	Variables, Scope, and Storage Duration	14
6	Expressions, Operators, and Evaluation	16
7	Strings, Numbers, Dates, and Exact Decimal	18
PART II	DATA, CONTROL, AND ABSTRACTION	20
8	Arrays	21
9	Maps and JSON	23
10	Conditional Control Structures	25
11	Loops and Iteration	27
12	Functions, Procedures, Parameters, and Recursion	29
13	Error Handling and Deterministic Cleanup	31
14	Classes and Objects	33
15	Codeblocks and Closures	35
PART III	MULTI-FILE PROGRAMS AND RUNTIME RESOURCES	37
16	Modules, Imports, Exports, and Separate Function Files	38
17	Project Manifests, Lockfiles, Bundles, and Standalone Builds	40
18	Text File Input and Output	42
19	Runtime Memory and Garbage Collection	44
PART IV	DATABASES, NETWORKING, AND CONCURRENCY	46
20	Database Programming	47
21	WebSocket Client Programming	50
22	Async Functions and Tasks	52
23	Named Threads, Dependencies, and Worker Pool	54
24	Safe Shared State: Mutexes, Atomics, and Channels	56
PART V	BACKEND SERVICES AND DEPLOYMENT	58
25	Writing HTTP Microservices	59
26	Deployment, Configuration, Security, and Testing	62
APPENDIX A	Core Syntax Quick Reference	64
APPENDIX B	Core Built-in Function Quick Reference	66
APPENDIX C	Files, Commands, and Artifact Reference	67
APPENDIX D	Verification Matrix and Troubleshooting	68

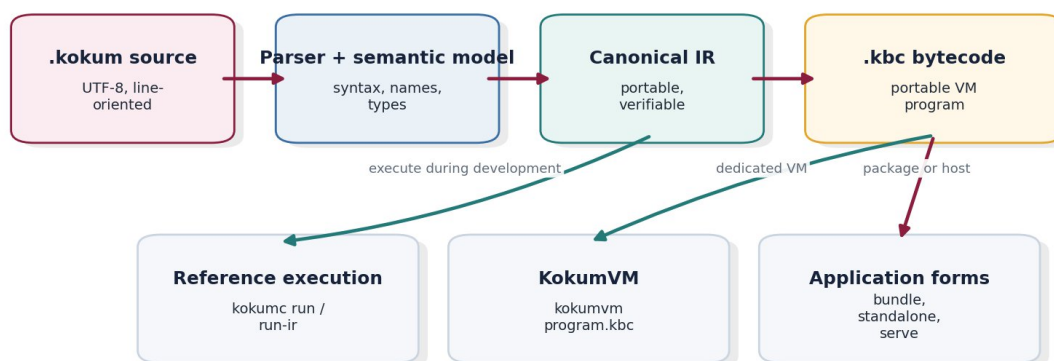
PART I

Language Foundations

Source structure, executable forms, types, storage, operators, and exact computation.

From Kokum source to execution

The same program can be checked, interpreted, compiled, bundled, or hosted as a service.



Kokum 0.29.0 core-language architecture

CHAPTER 01

1. First Program and Execution Model

A Kokum program is ordinary UTF-8 source. The compiler can validate and interpret it directly, or emit portable bytecode for KokumVM.

IN THIS CHAPTER

Learning outcomes

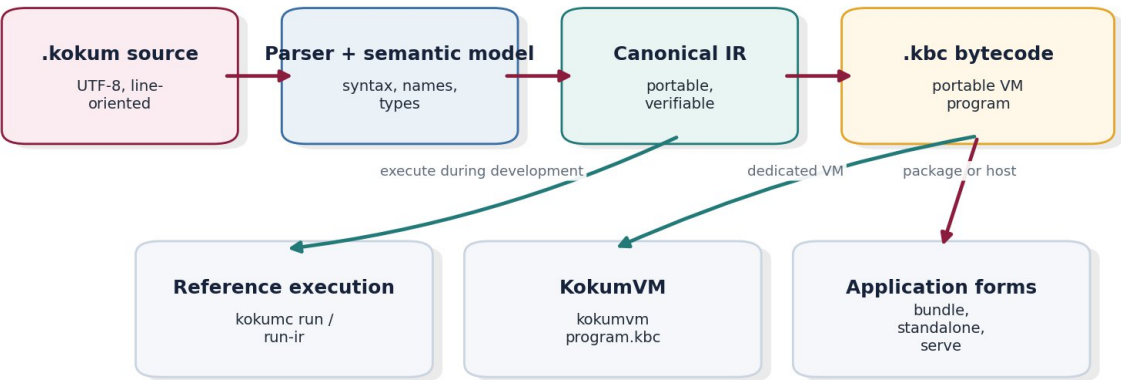
recognize the `Main` entry routine; run semantic checks; execute source and bytecode; understand the compiler-to-VM pipeline.

1.1 The minimum executable unit

A console program normally exposes a zero-argument `FUNCTION Main AS INTEGER`. The return value is the process result: `0` conventionally means success. `ENDFUNC` closes the declaration, and the `?` statement prints values followed by a newline.

From Kokum source to execution

The same program can be checked, interpreted, compiled, bundled, or hosted as a service.



Kokum 0.29.0 core-language architecture

Figure 1. One Kokum source file can be checked, interpreted, compiled, bundled, or hosted.

Listing 1. A complete first program - 01_program_structure.kokum

VERIFIED

A Kokum program begins at Main.
FUNCTION Main AS INTEGER
LOCAL language AS STRING

language = "Kokum"
? "Hello from " + language
RETURN 0
ENDFUNC

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

Hello from Kokum

1.2 The four-command development cycle

Listing 2. Check, interpret, compile, and execute

VERIFIED

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

Verification: Commands exercised against the Phase 4.16.7 binaries

Use **check** in editors and continuous integration because it performs parsing, name resolution, type checking, and semantic validation without executing the program. Use **run** for a quick development loop. Use **compile** plus **kokumvm** when validating the portable deployment path.

DESIGN RULE Keep Main small

Treat **Main** as orchestration: read configuration, call application functions, report fatal startup errors, and return a meaningful status. Put reusable work in functions, procedures, classes, or imported modules.

Chapter summary

- Kokum source uses the **.kokum** extension and normally begins execution at **Main**.
- The compiler supports semantic checking, source execution, IR execution, and portable bytecode emission.
- KokumVM runs **.kbc** modules and **.kapp** application bundles.

PRACTICE Try this

Change the first program so that **Main** calls a typed **Greeting(name AS STRING)** function and returns its result code.

CHAPTER 02

2. Source Structure, Comments, and Lexical Rules

Kokum keeps an XBase-style line-oriented surface while enforcing modern declaration and type rules.

IN THIS CHAPTER Learning outcomes

write valid line-oriented statements; continue a statement with semicolon; use all supported comment forms; apply identifier case rules safely.

2.1 Statements and continuation

A physical newline normally ends a statement. Put a semicolon at the end of a line to continue the same statement on the next line. Continuation is especially useful for long calls, collection literals, SQL strings, and response maps.

Form	Meaning	Example
Line-oriented statement	newline terminates the statement	<code>total = price + tax</code>
Continuation	semicolon joins the next physical line	<code>total = 10 + ;</code>
Print	question mark prints one or more values	<code>? "Total", total</code>
Routine terminator	declaration has an explicit closing keyword	<code>ENDFUNC, ENDPROC, ENDMETHOD</code>

2.2 Comments

Marker	Where it applies
<code>&&</code>	whole-line or trailing comment
<code>//</code>	whole-line or trailing comment
<code>--</code>	whole-line or trailing comment
<code>*</code>	comment when it is the first non-space character
<code>#</code>	comment when it is the first non-space character; otherwise <code>#</code> means not equal

2.3 Case-insensitive identifiers

Keywords and identifiers use case-insensitive ASCII canonicalization. `Total`, `total`, and `TOTAL` identify the same declaration. Kokum still retains the original spelling for diagnostics and reflection, so consistent casing remains a valuable style convention.

Listing 3. Comments, case-insensitivity, and continuation - 02_lexical_rules.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL Total AS INTEGER

  && Identifiers and keywords are case-insensitive.
  total = 10 + ;
    20 + ;   && Semicolon continues the statement.
    30

  ? "TOTAL =", TOTAL
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output
TOTAL = 60

COMPATIBILITY NOTE Reserved lifecycle words

SHOW, **HIDE**, **CLOSE**, and **DISPOSE** are language statements. Do not reuse them as routine names; choose names such as **DisplayMessage** or **RenderReport**.

Chapter summary

- A newline terminates a statement unless the previous physical line ends with ;.
- Four comment styles are available, with special rules for * and #.
- Identifiers are case-insensitive, but consistent spelling improves readability.

PRACTICE Try this

Rewrite the continued arithmetic expression as a continued function call with three arguments and one trailing comment.

CHAPTER 03

3. Kokum Files, Projects, and Build Artifacts

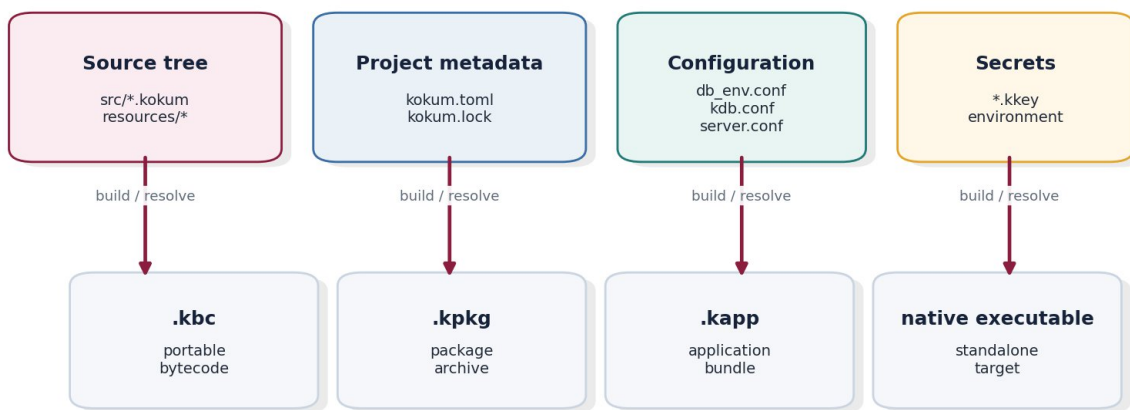
A Kokum application is more than source files: the toolchain uses a small set of explicit metadata, configuration, package, bytecode, and deployment formats.

IN THIS CHAPTER Learning outcomes

identify source and generated files; separate configuration from code; choose `.kbc`, `.kapp`, or standalone output; organize a reproducible project tree.

Kokum project files and generated artifacts

Source and configuration remain readable; generated forms are portable and reproducible.



Kokum 0.29.0 core-language architecture

Figure 2. Source, project metadata, configuration, secrets, and generated artifacts have distinct roles.

File or suffix	Purpose	Normally committed?
*.kokum	Kokum source module or standalone program	Yes
kokum.toml	strict project manifest: identity, entry point, modules, resources, build and bundle options	Yes
kokum.lock	resolved module/package versions and identities	Yes for applications
*.kbc	portable compiled bytecode module	Build output
*.kpkg	versioned package archive	Published artifact
*.kapp	portable application bundle containing modules, resources, packages, and optional native payloads	Deployment artifact
server.conf	HTTP service host and route configuration	Template yes; environment-specific values reviewed
db_env.conf	database provider configuration and safe substitutions	Usually no; publish an example template
kdb.conf	Remote FlatDB endpoint and key-file location	Environment-specific
*.kkey	binary Remote FlatDB authentication key	Never commit production keys
native executable	standalone launcher/application output	Release artifact

3.1 A small artifact probe

Listing 4. Source used to demonstrate generated forms - 22_artifact_probe.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL format AS STRING

  format = "portable bytecode"
  ? "Kokum emits " + format
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

Kokum emits portable bytecode

3.2 Generated forms

Listing 5. Typical artifact commands

VERIFIED

```
./kokumc compile 22_artifact_probe.kokum -o artifact_probe.kbc
./kokumvm artifact_probe.kbc

# In a project:
./kokumc project build kokum.toml
./kokumc project bundle kokum.toml
./kokumc project build kokum.toml --standalone dist/artifact_probe
```

Verification: Standalone and project artifact paths were exercised in this edition

SECURITY Configuration is not the same as a secret

A readable configuration file can name a credential environment variable or key-file path. It should not contain production passwords, bearer tokens, private keys, or Remote FlatDB key bytes in source control.

Chapter summary

- Source, project metadata, configuration, secrets, and generated artifacts should remain separate.
- `.kbc` is a portable VM module; `.kapp` is a deployable application bundle.
- Lockfiles make dependency resolution repeatable.

PRACTICE Try this

Create a project folder with `src`, `config`, and `dist` directories. Decide which files belong in version control and explain why.

CHAPTER 04

4. Types and Literals

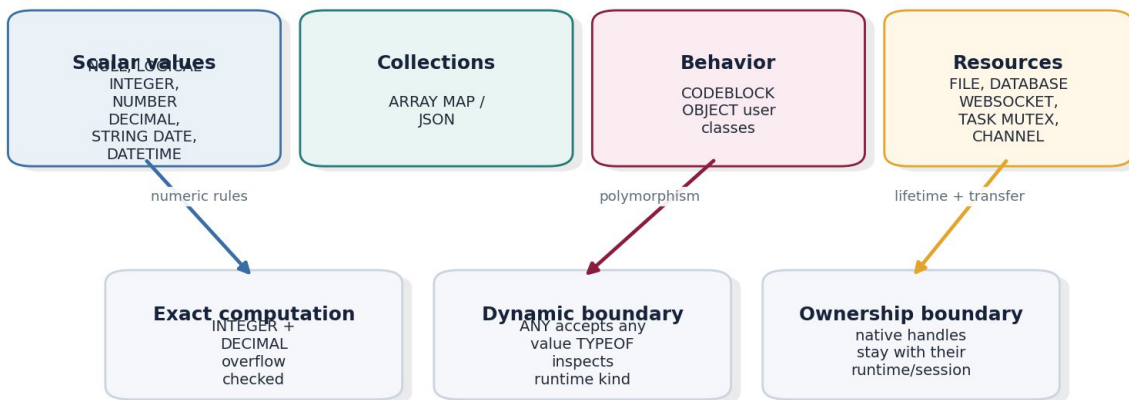
Kokum combines strict declarations with an `ANY` escape hatch. Scalar, collection, behavior, and resource values obey different conversion and ownership rules.

IN THIS CHAPTER Learning outcomes

declare core types; choose integer, binary number, or exact decimal; use **NULL** and **ANY** deliberately; inspect runtime kinds with **TYPEOF**.

Kokum type families

Annotations describe ordinary values, collections, behavior, and owner-runtime resources.



Kokum 0.29.0 core-language architecture

Figure 3. Core types fall into scalar, collection, behavior, and owner-runtime resource families.

Type	Use	Representative literal or creator
NULL	absence of a value	NULL
LOGICAL	boolean state	.T. , .F.
INTEGER	signed 64-bit whole number	42
NUMBER	binary double-precision number	2.5
DECIMAL / MONEY	exact base-10 calculation	199.95M , DECIMAL ("199.95")
STRING	UTF-8 text	"Kokum"
DATE	calendar date	DATE()
DATETIME	date and time	DATETIME()
ARRAY	mutable ordered sequence	[10, 20]
MAP / JSON	mutable insertion-ordered key/value data	{"name": "Kokum"}
CODEBLOCK	captured expression behavior	{ x x * 2}
OBJECT / class name	class instance	NEW Account(...)
Resource types	FILE, DATABASE, WEBSOCKET, TASK, MUTEX, ATOMICINTEGER, CHANNEL	created or supplied by the relevant runtime API
ANY	value whose runtime kind may vary	assignment from any supported value

Listing 6. Core type declarations and runtime inspection - 03_types_and_literals.kokum

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL count AS INTEGER
  LOCAL ratio AS NUMBER
  LOCAL amount AS DECIMAL
  LOCAL enabled AS LOGICAL
  LOCAL title AS STRING
  LOCAL today AS DATE
  LOCAL now AS DATETIME
  LOCAL missing AS ANY

  count = 42
  ratio = 2.5
  amount = 199.95M
  enabled = .T.
  title = "Core language"
  today = DATE()
  now = DATETIME()
  missing = NULL

  ? TYPEOF(count), TYPEOF(ratio), TYPEOF(amount)
  ? TYPEOF(enabled), TYPEOF(title)
  ? TYPEOF(today), TYPEOF(now), TYPEOF(missing)
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

INTEGER NUMBER DECIMAL
LOGICAL STRING
DATE DATETIME NULL

```

4.1 Assignment compatibility

A declared variable accepts values compatible with its annotation. **ANY** is useful at a data boundary, but code should narrow the value with **TYPEOF**, validation, or an explicit conversion before performing type-specific operations. A strongly typed variable should not be reused for an unrelated kind.

NUMERIC RULE Binary and exact arithmetic are intentionally distinct

INTEGER and **DECIMAL** may mix exactly. **DECIMAL** and binary **NUMBER** do not mix implicitly; convert deliberately so a financial calculation cannot silently become binary floating point.

Chapter summary

- Use the most specific annotation that represents the value.
- Choose **DECIMAL** for exact base-10 work such as prices, tax, and accounting.
- Resource values have ownership and transfer restrictions beyond ordinary scalar values.

PRACTICE Try this

Add an **ARRAY**, a **MAP**, and a **CODEBLOCK** to the example. Print **TYPEOF** for each and explain the result.

CHAPTER 05

5. Variables, Scope, and Storage Duration

Kokum requires declarations before use. Storage keywords express visibility and lifetime rather than creating values implicitly on assignment.

IN THIS CHAPTER Learning outcomes

distinguish LOCAL, PRIVATE, STATIC, and PUBLIC; understand initialization and lifetime; avoid implicit-variable assumptions; use typed parameters.

Storage classes and lifetime

Choose the narrowest lifetime that correctly represents the data.

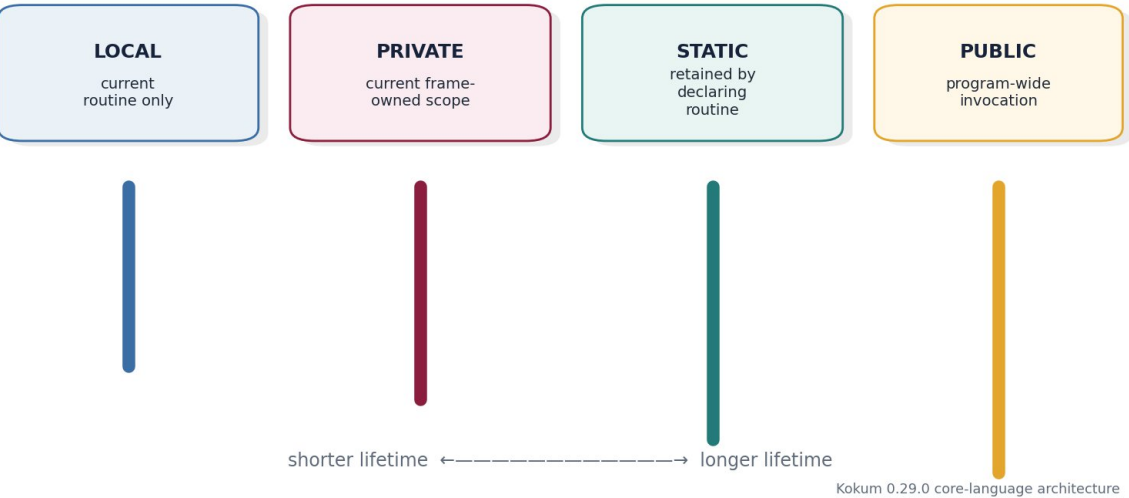


Figure 4. Storage classes move from routine-local state to program-wide state.

Declaration	Visibility and lifetime	Typical use
LOCAL	current routine or method invocation	temporary calculations, loop variables, local resources
PRIVATE	owned by the current frame in the modern core	frame-scoped compatibility state
STATIC	retained for the declaring routine; method statics are class-qualified	sequence generators, cached immutable data
PUBLIC	program-wide for one interpreter/VM invocation	explicit application resources and shared configuration handles
PARAMETERS / LPARAMETERS	routine input declarations	typed function/procedure parameters

Listing 7. Public, private, local, and static storage - 04_variables_and_scope.kokum

VERIFIED

```
PUBLIC applicationName AS STRING

FUNCTION NextNumber AS INTEGER
  STATIC sequence AS INTEGER

  IF sequence = NULL
    sequence = 0
  ENDIF
  sequence = sequence + 1
  RETURN sequence
ENDFUNC

FUNCTION Main AS INTEGER
```

```
LOCAL first AS INTEGER
PRIVATE scratch AS STRING

applicationName = "Kokum"
scratch = "temporary"
first = NextNumber()

? applicationName, scratch
? first, NextNumber(), NextNumber()
RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
Kokum temporary
1 2 3
```

5.1 Initialization and NULL

A declaration creates a variable slot; its initial state can be **NULL** until assigned. Test for **NULL** when a routine-local **STATIC** must be initialized once. Assignment never creates a new variable implicitly, so misspellings become semantic errors instead of hidden state.

STYLE Prefer narrow scope

Use **LOCAL** by default. Introduce **STATIC** only when persistence is part of the routine contract and **PUBLIC** only for a clearly named application-wide resource. Narrow scope makes concurrent and service code easier to reason about.

Chapter summary

- Variables must be declared before use.
- **STATIC** persists between calls; **PUBLIC** is visible program-wide during one invocation.
- Assignment does not manufacture undeclared variables.

PRACTICE Try this

Change **NextNumber** into **NextNumber(step AS INTEGER)** and verify that the static sequence advances by the supplied step.

CHAPTER 06

6. Expressions, Operators, and Evaluation

Expressions combine literals, variables, calls, indexes, members, and operators. Kokum evaluates left to right and short-circuits logical operators.

IN THIS CHAPTER Learning outcomes

apply arithmetic, comparison, and logical operators; understand equality spellings; use grouping and short-circuit logic; anticipate checked integer behavior.

Category	Operators	Notes
Arithmetic	<code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>%</code>	integer overflow is checked; division behavior follows operand types
Equality	<code>=</code> <code>==</code>	both express equality in expression context
Inequality	<code>!=</code> <code><></code> <code>#</code>	<code>#</code> is an operator unless it begins a comment line
Ordering	<code><</code> <code><=</code> <code>></code> <code>>=</code>	operands must support the comparison
Logical	<code>AND</code> / <code>.AND.</code> , <code>OR</code> / <code>.OR.</code> , <code>NOT</code> / <code>.NOT.</code>	conditions are logical and AND/OR short-circuit
Grouping	<code>(expression)</code>	makes precedence and intent explicit
Access	<code>array[index]</code> , <code>map[key]</code> , <code>value.member</code>	array indexes are one-based

Listing 8. Arithmetic, comparison, and logical expressions - `05_expressions_and_operators.kokum`

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL a AS INTEGER
  LOCAL b AS INTEGER
  LOCAL allowed AS LOGICAL

  a = 17
  b = 5
  allowed = a > 10 AND b < 10 AND NOT (a = b)

  ? "sum", a + b
  ? "difference", a - b
  ? "product", a * b
  ? "division", a / b
  ? "remainder", a % b
  ? "allowed", allowed
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

sum 22
difference 12
product 85
division 3.4
remainder 2
allowed .T.

```

6.1 Short-circuiting as a safety tool

In `ready AND ExpensiveCheck()`, the call is skipped when `ready` is false. In `cached OR LoadValue()`, the load is skipped when `cached` is true. Use this behavior for guarded access, but do not hide essential side effects inside a boolean expression.

RELIABILITY Make conversions visible

When data crosses a text, JSON, database, or network boundary, convert explicitly with functions such as `VAL`,

DECIMAL, or **STR**. Visible conversion points simplify validation and error handling.

Chapter summary

- Evaluation is left to right; AND and OR short-circuit.
- Multiple equality and inequality spellings support familiar XBase styles.
- Integer overflow and invalid mixed numeric operations raise errors rather than silently corrupting data.

PRACTICE Try this

Write a condition that calls a validation function only when a value is non-NULL. Explain which operand prevents the call.

CHAPTER 07

7. Strings, Numbers, Dates, and Exact Decimal

Text conversion and numeric representation are core design choices. Kokum separates binary floating point from exact decimal arithmetic.

IN THIS CHAPTER Learning outcomes

use common string functions; parse and format numbers; work with DATE and DATETIME values; perform exact decimal calculations.

7.1 Text and numeric conversion

Listing 9. Text casing, length, parsing, and formatting - 06_strings_and_numbers.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL text AS STRING
  LOCAL parsed AS NUMBER

  text = "Kokum Language"
  parsed = VAL("125.50")

  ? UPPER(text)
  ? LOWER(text)
  ? "length", LEN(text)
  ? "parsed + 4.5 =", parsed + 4.5
  ? "formatted", STR(parsed, 10, 2)
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
KOKUM LANGUAGE
kokum language
length 14
parsed + 4.5 = 130
formatted      125.50
```

Function	Purpose
<code>LEN(value)</code>	string length or collection size
<code>UPPER(text)</code> / <code>LOWER(text)</code>	ASCII-oriented case conversion for ordinary source strings
<code>VAL(text)</code>	parse a binary numeric value
<code>DECIMAL(text)</code>	parse an exact decimal value
<code>STR(value [, width [, decimals]])</code>	format a value as text
<code>DATE()</code>	current date value
<code>DATETIME()</code>	current date/time value
<code>typeof(value)</code>	runtime kind name

7.2 Exact decimal arithmetic

A decimal literal ends in **M**. **DECIMAL** stores exact base-10 digits under a 4,096-digit safety bound. This avoids binary representation surprises for money, percentages, quantities, and financial totals. Decimal division produces a bounded exact result with defined rounding behavior.

Listing 10. Exact tax calculation - 07_exact_decimals.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL price AS DECIMAL
```

```
LOCAL taxRate AS DECIMAL
LOCAL tax AS DECIMAL
LOCAL total AS DECIMAL

price = 199.95M
taxRate = DECIMAL("18.00")
tax = price * taxRate / 100M
total = price + tax

? "Price:", STR(price, 10, 2)
? "Tax: ", STR(tax, 10, 2)
? "Total:", STR(total, 10, 2)
RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
Price:      199.95
Tax:        35.99
Total:      235.94
```

FINANCIAL PROGRAMMING Round at a business boundary

Keep intermediate values as **DECIMAL**. Format or round when a business rule requires a currency precision, invoice line policy, tax rule, or external schema. Do not convert to **NUMBER** merely for convenience.

Chapter summary

- Strings are UTF-8 sequences and have explicit conversion helpers.
- **NUMBER** is binary floating point; **DECIMAL** is exact base-10.
- Use **STR** for presentation and **VAL/DECIMAL** for validated input conversion.

PRACTICE Try this

Calculate a 12.5 percent discount on **799.90M**, print the discount and final price with two decimal places, and compare the result with a binary **NUMBER** implementation.

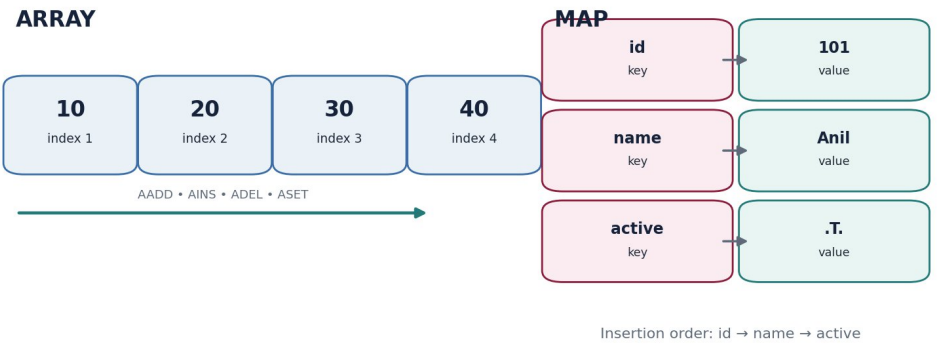
PART II

Data, Control, and Abstraction

Collections, decisions, loops, routines, errors, classes, and executable codeblocks.

Arrays are one-based; maps preserve insertion order

Iteration snapshots the collection at loop entry.



Kokum 0.29.0 core-language architecture

CHAPTER 08

8. Arrays

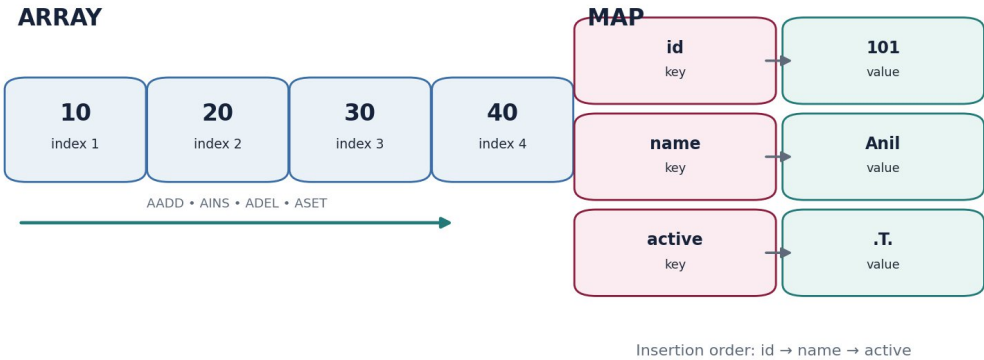
Arrays are mutable, ordered sequences with one-based source indexes. Their operations make insertion, replacement, deletion, and traversal explicit.

IN THIS CHAPTER Learning outcomes

create and index arrays; modify arrays safely; iterate with values or index/value pairs; handle one-based boundaries.

Arrays are one-based; maps preserve insertion order

Iteration snapshots the collection at loop entry.



Kokum 0.29.0 core-language architecture

Figure 5. Arrays use one-based indexes; maps retain insertion order.

Operation	Behavior
<code>array[index]</code>	read or assign at a one-based index
<code>AADD(array, value)</code>	append and return the value
<code>ASET(array, index, value)</code>	replace and return the value
<code>AINS(array, index, value)</code>	insert before index; <code>LEN(array)+1</code> appends
<code>ADEL(array, index)</code>	remove and return the old value
<code>LEN(array)</code>	current element count

Listing 11. One-based array mutation and iteration - 08_arrays.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL numbers AS ARRAY
  LOCAL index AS INTEGER
  LOCAL value AS INTEGER

  numbers = [10, 20, 30]
  numbers[2] = 25
  AADD(numbers, 40)
  AINS(numbers, 2, 15)

  FOR EACH index, value IN numbers
    ? index, value
  NEXT

  ? "removed", ADEL(numbers, 1)
  ? JSONENCODE(numbers)
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
1 10
2 15
3 25
4 30
5 40
removed 10
[15,25,30,40]
```

8.1 Boundary behavior

Valid ordinary indexes are `1..LEN(array)`. Zero, negative, noninteger, and out-of-range indexes raise catchable errors. **AINS** additionally accepts `LEN(array)+1` so insertion can append. Use **FOR EACH index, value IN array** when the position matters.

ITERATION CONTRACT The loop sees a snapshot

Collection iteration snapshots the array at loop entry. Mutating the original collection does not unpredictably change the current traversal sequence.

Chapter summary

- Array source indexes begin at 1.
- Mutation functions return the affected value, which can simplify pipelines.
- **FOR EACH** supports value-only and index/value forms.

PRACTICE Try this

Create an array of prices, insert a new first value, remove the last value, and calculate the exact decimal total with **FOR EACH**.

CHAPTER 09

9. Maps and JSON

Maps provide mutable insertion-ordered key/value storage. JSON conversion makes maps and arrays natural boundary types for databases, WebSockets, and HTTP services.

IN THIS CHAPTER Learning outcomes

create and update maps; use bracket and dot access; distinguish missing keys with **HASKEY**; encode and decode JSON.

Operation	Behavior
<code>map[key]</code>	read or assign using any supported key value
<code>map.member</code>	string-key shortcut for a valid member name
<code>MAPSET(map, key, value)</code>	set and return value
<code>HASKEY(map, key)</code>	test membership explicitly
<code>MAPREMOVE(map, key)</code>	remove and return old value or NULL
<code>KEYS(map) / VALUES(map)</code>	new arrays in insertion order
<code>JSONENCODE(value)</code>	canonical JSON text for JSON-safe data
<code>JSONDECODE(text)</code>	parse JSON to Kokum values

Listing 12. Insertion-ordered maps and JSON round-trip - 09_maps_and_json.kokum

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL customer AS MAP
  LOCAL key
  LOCAL value
  LOCAL decoded AS JSON

  customer = {"id": 101, "name": "Anil"}
  customer.active = .T.
  customer["city"] = "Ratnagiri"

  FOR EACH key, value IN customer
    ? key, value
  NEXT

  decoded = JSONDECODE(JSONENCODE(customer))
  ? "has city", HASKEY(decoded, "city")
  ? JSONENCODE(decoded)
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

id 101
name Anil
active .T.
city Ratnagiri
has city .T.
{"id":101,"name":"Anil","active":true,"city":"Ratnagiri"}

```

9.1 Missing keys and NULL

Reading a missing map key returns **NULL**. This is convenient for optional fields, but it does not distinguish an absent key from a present key whose value is **NULL**. Use **HASKEY** when the distinction matters.

BOUNDARY PATTERN Validate decoded data before trusting it

After **JSONDECODE**, inspect the value kind, required keys, field types, and permitted ranges before using it in SQL,

thread scheduling, file paths, or service responses.

Chapter summary

- Maps preserve insertion order and support bracket or member-style string keys.
- Missing reads return **NULL**; **HASKEY** tests actual membership.
- Arrays and maps are the principal structured JSON values.

PRACTICE Try this

Decode a JSON customer record, require **id** and **name**, add a computed **active** field, and encode the result again.

CHAPTER 10

10. Conditional Control Structures

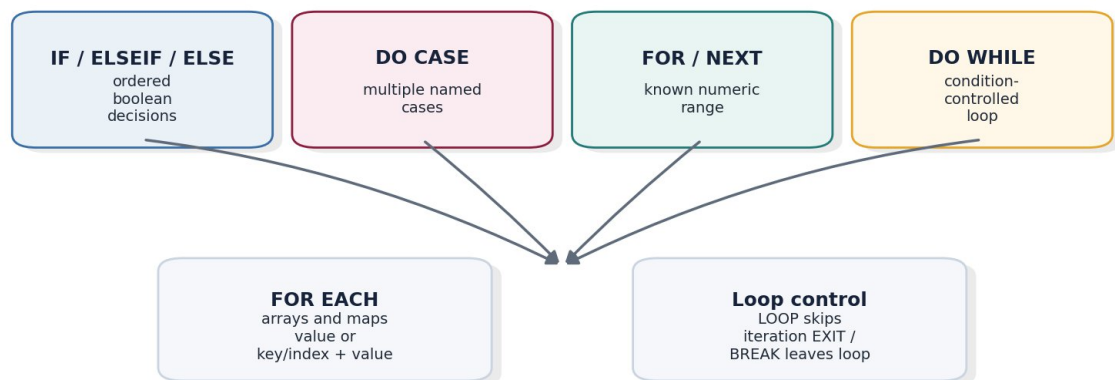
Kokum offers a conventional IF chain and an XBase-style DO CASE construct. Both require logical conditions and explicit terminators.

IN THIS CHAPTER Learning outcomes

write IF/ELSEIF/ELSE decisions; express ordered cases with DO CASE; return early from functions; avoid overlapping branch ambiguity.

Control-flow choices

Use the construct that makes the decision or repetition easiest to read.



Kokum 0.29.0 core-language architecture

Figure 6. Choose a decision or loop construct that exposes intent clearly.

10.1 IF chains

Use **IF** when the code has one primary condition or a short ordered chain. Conditions are logical; do not rely on implicit truthiness of numbers, strings, or collections.

10.2 DO CASE

DO CASE reads well when several named alternatives compete in priority order. The first true **CASE** executes. **OTHERWISE** handles the remainder, and **ENDCASE** closes the structure.

Listing 13. IF and DO CASE decisions - 10_decisions.kokum

VERIFIED

```

FUNCTION RiskBand(score AS INTEGER) AS STRING
  IF score >= 80
    RETURN "HIGH"
  ELSEIF score >= 50
    RETURN "MEDIUM"
  ELSE
    RETURN "LOW"
  ENDIF
ENDFUNC

FUNCTION Main AS INTEGER
  LOCAL score AS INTEGER

  score = 72
  ? "IF result:", RiskBand(score)

  DO CASE
    CASE score >= 90
      ? "Excellent"
  
```

```
CASE score >= 60
  ? "Good"
OTHERWISE
  ? "Needs review"
ENDCASE

RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
IF result: MEDIUM
Good
```

DESIGN RULE Order ranges from most specific to least specific

When thresholds overlap, place the strongest condition first. A return from a matched branch often makes the remaining branches easier to read and test.

Chapter summary

- Conditions must evaluate to **LOGICAL**.
- **ELSEIF** builds an ordered chain; **DO CASE** emphasizes named alternatives.
- The first satisfied branch wins.

PRACTICE Try this

Write a **DO CASE** function that classifies an HTTP status into informational, success, redirect, client error, or server error.

CHAPTER 11

11. Loops and Iteration

Numeric loops, condition loops, and collection iteration cover most repetition patterns. `LOOP` and `EXIT` make exceptional flow explicit.

IN THIS CHAPTER Learning outcomes

use FOR with optional STEP; write DO WHILE loops; iterate arrays and maps; control loops with LOOP and EXIT.

Construct	Best use
FOR i = start TO finish [STEP step] ... NEXT	known numeric range
DO WHILE condition ... ENDDO	repeat while a changing condition remains true
FOR EACH value IN collection ... NEXT	consume array or map values
FOR EACH key, value IN collection ... NEXT	also receive one-based array index or map key
LOOP	skip the rest of this iteration
EXIT / BREAK	leave the nearest loop

Listing 14. Numeric, condition, and collection loops - 11_loops.kokum (continued 1/2)

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL i AS INTEGER
  LOCAL total AS INTEGER
  LOCAL values AS ARRAY
  LOCAL value AS INTEGER

  total = 0
  FOR i = 1 TO 5
    total = total + i
  NEXT
  ? "FOR total", total

  i = 1
  DO WHILE i <= 3
    ? "WHILE", i
    i = i + 1
  ENDDO

  values = [2, 4, 6, 8]
  FOR EACH value IN values
    IF value = 4
      LOOP
    ENDIF
    IF value = 8
      EXIT
    ENDIF
    ? "EACH", value
  NEXT

  RETURN 0

```

Listing 14. Numeric, condition, and collection loops - 11_loops.kokum (continued 2/2)

VERIFIED

```

ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

FOR total 15
WHILE 1
WHILE 2
WHILE 3
EACH 2
EACH 6

```

11.1 Termination and progress

Every **DO WHILE** loop should have an obvious progress operation or an explicit blocking operation with a timeout. In network, channel, or service code, always decide how the loop terminates after EOF, close, error, cancellation by process shutdown, or a timeout.

REVIEW QUESTION Can this loop run forever?

Before release, review every unbounded loop for a clear stop condition, bounded wait, resource cleanup path, and diagnostic behavior.

Chapter summary

- **FOR** handles ranges; **DO WHILE** handles evolving conditions; **FOR EACH** handles collections.
- **LOOP** continues with the next iteration and **EXIT** leaves the loop.
- Collection iteration uses an entry snapshot.

PRACTICE Try this

Use **FOR EACH** to sum only positive values. Skip zero with **LOOP** and stop when a negative sentinel appears.

CHAPTER 12

12. Functions, Procedures, Parameters, and Recursion

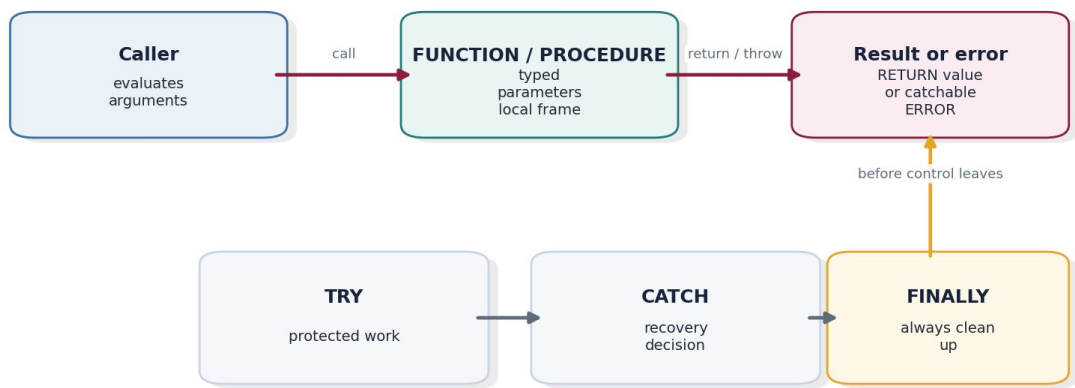
Routines are the primary units of behavior. Typed declarations expose contracts; recursion expresses naturally recursive problems when depth is bounded.

IN THIS CHAPTER Learning outcomes

declare functions and procedures; use inline and LPARAMETERS forms; return typed results; write and review recursive routines.

Routine calls, returns, and deterministic cleanup

FINALLY runs on success, error propagation, loop control, and RETURN.



Kokum 0.29.0 core-language architecture

Figure 7. A routine call creates a frame and returns a value or propagates an error; FINALLY still executes.

Routine form	Result
FUNCTION Name(...) [AS Type]	returns a value; unannotated functions remain dynamically typed
PROCEDURE Name(...)	performs work; successful completion produces no meaningful value
METHOD Name(...) [AS Type]	class behavior with implicit THIS
ASYNC FUNCTION/PROCEDURE/METHOD	returns a TASK immediately
PARAMETERS / LPARAMETERS	alternative parameter declaration inside a routine

Listing 15. Typed routines and LPARAMETERS - 12_functions_and_procedures.kokum

VERIFIED

```
FUNCTION Add(left AS INTEGER, right AS INTEGER) AS INTEGER
  RETURN left + right
ENDFUNC

FUNCTION Describe
  LPARAMETERS name AS STRING, count AS INTEGER
  RETURN name + ": " + STR(count)
ENDFUNC

PROCEDURE DisplayMessage(message AS STRING)
  ? message
ENDPROC

FUNCTION Main AS INTEGER
  DisplayMessage(Describe("items", Add(2, 3)))
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

items: 5

12.1 Recursion

Listing 16. A bounded recursive factorial - 13_recursion.kokum

VERIFIED

```
FUNCTION Factorial(n AS INTEGER) AS INTEGER
  IF n <= 1
    RETURN 1
  ENDIF
  RETURN n * Factorial(n - 1)
ENDFUNC

FUNCTION Main AS INTEGER
  ? "6! =", Factorial(6)
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

6! = 720

A recursive function needs a base case and a progress step that moves every call toward that base case. For untrusted or very deep input, prefer an iterative algorithm or impose a validated depth limit.

API DESIGN Make the contract visible

Use descriptive names, typed parameters, one clear return type, and small routines. Validate boundary data before delegating to a reusable core function.

Chapter summary

- Functions return values; procedures communicate through effects and explicit resources.
- Parameters can be declared in the signature or through **LPARAMETERS**.
- Recursion must have a reachable base case and bounded depth.

PRACTICE Try this

Write an iterative and a recursive function for the same small problem. Compare readability, failure modes, and maximum input size.

CHAPTER 13

13. Error Handling and Deterministic Cleanup

TRY/CATCH/FINALLY separates normal work, recovery, and unconditional cleanup. The FINALLY block runs even when control returns or leaves a loop.

IN THIS CHAPTER Learning outcomes

catch runtime errors; return fallback values deliberately; guarantee cleanup with FINALLY; avoid suppressing diagnostic context.

13.1 Structured error flow

Put only the operations that share one recovery policy inside a TRY. CATCH error receives an ERROR value. FINALLY executes on successful completion, error propagation, loop control, and return. A new nonnormal flow from FINALLY replaces the pending flow, so FINALLY should usually perform cleanup rather than return or throw.

Listing 17. Catch an arithmetic failure and run cleanup - 14_error_handling.kokum

VERIFIED

```
FUNCTION SafeDivide(value AS INTEGER) AS NUMBER
  TRY
    RETURN 100 / value
  CATCH error
    ? "Caught:", TYPEOF(error)
    RETURN 0
  FINALLY
    ? "SafeDivide cleanup"
  ENDTRY
ENDFUNC

FUNCTION Main AS INTEGER
  ? "Result:", SafeDivide(0)
  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```
Caught: ERROR
SafeDivide cleanup
Result: 0
```

Layer	Responsibility
Core calculation	raise or propagate an error when its contract cannot be met
Boundary routine	translate expected failures to a result, status, or diagnostic appropriate for the caller
FINALLY	close FILE values, rollback unfinished transactions, release locks, or restore state
Main/service boundary	log enough context and return a meaningful process or HTTP result

RELIABILITY Do not catch everything without a policy

A catch block that prints a generic message and continues can hide corrupted state. Recover only when the routine can restore a valid invariant; otherwise add context and allow the error to propagate.

Chapter summary

- CATCH handles a runtime error value; FINALLY always runs during control transfer.
- Cleanup belongs in FINALLY, not duplicated across success and failure branches.
- Recovery must restore a valid state or propagate the failure.

PRACTICE Try this

Wrap the file example from Chapter 18 in TRY/CATCH/FINALLY and verify that CLOSEFILE is reached after a deliberate invalid path error.

CHAPTER 14

14. Classes and Objects

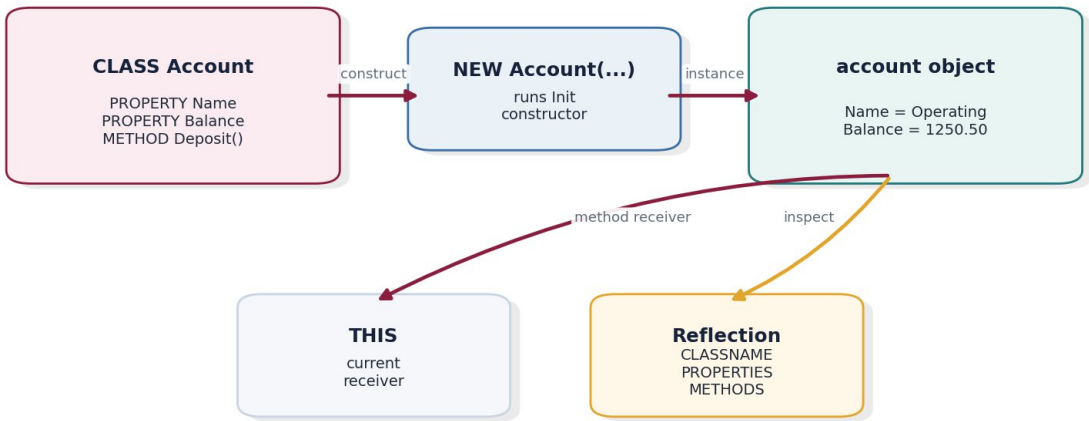
Classes combine properties and methods with optional single inheritance. `NEW` creates an instance and invokes the effective `Init` constructor.

IN THIS CHAPTER Learning outcomes

declare properties and methods; construct objects with NEW; use THIS correctly; inspect objects through reflection.

Classes, objects, properties, and methods

A class defines state and behavior; each NEW call creates an independent object.



Kokum 0.29.0 core-language architecture

Figure 8. A class defines state and behavior; NEW constructs independent instances.

Feature	Syntax or behavior
Class declaration	CLASS Name [AS BaseClass] ... ENDClass
Property	PROPERTY Name [AS Type] [= default]
Method	METHOD Name(...) [AS Type] ... ENDMETHOD
Current receiver	THIS
Construction	NEW ClassName(arguments) invokes Init when present
Inheritance	single base class; methods may override inherited methods
Reflection	CLASSNAME, PROPERTIES, METHODS, HASPROPERTY, HASMETHOD, GETPROPERTY, SETPROPERTY

Listing 18. Account class with constructor and method - 15_classes_and_objects.kokum

VERIFIED

```
CLASS Account
  PROPERTY Name AS STRING = "Unnamed"
  PROPERTY Balance AS DECIMAL = 0M

  METHOD Init(name AS STRING, opening AS DECIMAL)
    THIS.Name = name
    THIS.Balance = opening
  ENDMETHOD

  METHOD Deposit(amount AS DECIMAL) AS DECIMAL
    THIS.Balance = THIS.Balance + amount
```

```

        RETURN THIS.Balance
    ENDMETHOD
ENDCLASS

FUNCTION Main AS INTEGER
    LOCAL account AS Account

    account = NEW Account("Operating", 1000M)
    account.Deposit(250.50M)

    ? account.Name
    ? STR(account.Balance, 12, 2)
    ? CLASSNAME(account)
    RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

Operating
1250.50
Account

```

14.1 Invariants

Use **Init** to establish valid initial state. Methods should preserve class invariants such as nonnegative balances, permitted status transitions, or validated identifiers. Avoid exposing mutable internal collections unless the caller is intended to own them.

CURRENT BOUNDARY Inheritance checks continue to evolve

Method overriding is supported, but advanced nominal subtyping and full override-signature compatibility are not a substitute for explicit tests. Prefer small interfaces and composition when a hierarchy would be fragile.

Chapter summary

- Properties hold object state and methods operate through **THIS**.
- **NEW** creates an instance and invokes **Init** when defined.
- Reflection can inspect and manipulate public object members.

PRACTICE Try this

Add a **Withdraw** method that rejects a negative amount and prevents the balance from becoming negative. Test both success and failure paths.

CHAPTER 15

15. Codeblocks and Closures

A codeblock is a one-expression callable value. It can capture lexical variables, travel inside collections, and be invoked directly or through EVAL.

IN THIS CHAPTER Learning outcomes

write codeblock literals; capture lexical variables; store behavior in collections; invoke blocks directly and with EVAL.

Form	Meaning
<code>{ x x * 2}</code>	one-parameter expression block
<code>{ x, y x + y}</code>	two-parameter expression block
<code>block(value)</code>	direct invocation
<code>EVAL(block, value)</code>	built-in invocation form
Captured free variable	shared lexical cell retained by the closure

Listing 19. Returned closure and map-held behavior - 16_codeblocks_and_closures.kokum

VERIFIED

```

FUNCTION MakeMultiplier(factor AS INTEGER) AS CODEBLOCK
  RETURN {|value| value * factor}
ENDFUNC

FUNCTION Main AS INTEGER
  LOCAL doubleValue AS CODEBLOCK
  LOCAL operations AS MAP

  doubleValue = MakeMultiplier(2)
  operations = {"square": {|value| value * value}}

  ? doubleValue(21)
  ? EVAL(doubleValue, 12)
  ? operations.square(9)
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

42
24
81

```

The current core codeblock contains one expression. Use a named function when behavior needs statements, detailed validation, TRY/CATCH, multiple returns, or a public module contract.

CONCURRENCY A codeblock is not a thread message

Codeblocks capture runtime cells and are intentionally not transferable into isolated async workers or channel messages. Pass ordinary data and call a named routine inside the worker instead.

Chapter summary

- Codeblocks are typed as **CODEBLOCK** and contain one expression.
- Closures retain access to lexical free variables.
- Direct invocation and **EVAL** are equivalent execution choices.

PRACTICE Try this

Build a map of **double**, **triple**, and **square** codeblocks, choose one by key, and invoke it on a validated integer.

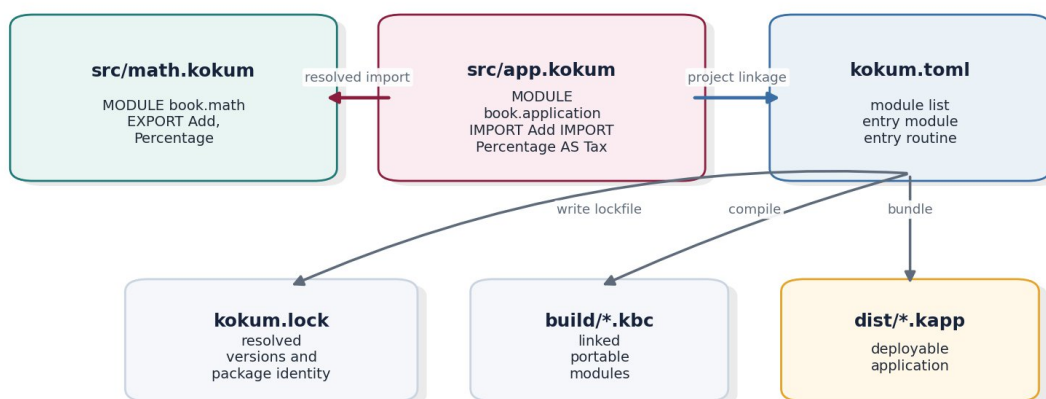
PART III

Multi-file Programs and Runtime Resources

Explicit modules, reproducible projects, files, bundles, and managed runtime memory.

Separate files become explicit modules

IMPORT resolves a declared and exported symbol under a version constraint.



Kokum 0.29.0 core-language architecture

CHAPTER 16

16. Modules, Imports, Exports, and Separate Function Files

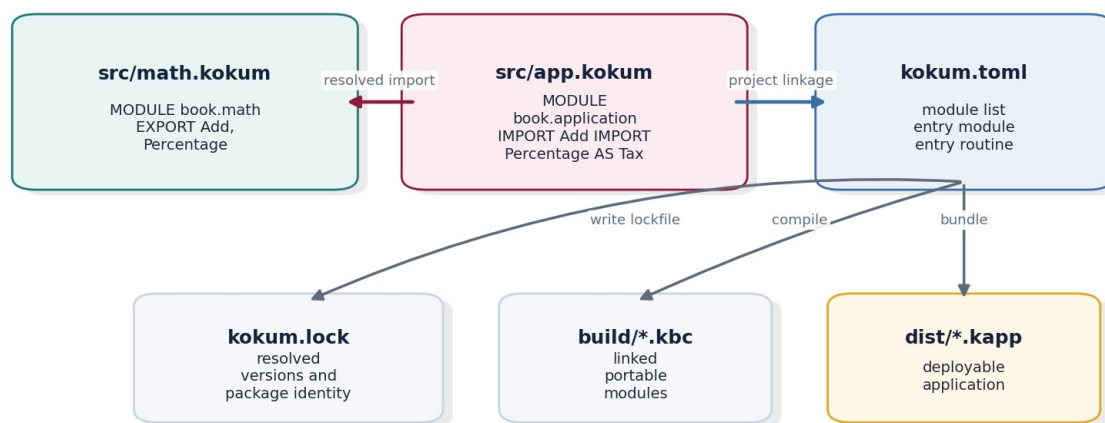
A module gives a source file a stable logical name and version. Exports define its public surface; imports resolve required symbols explicitly.

IN THIS CHAPTER Learning outcomes

place reusable functions in another file; declare module identity and version; export a public API; import with version constraints and aliases.

Separate files become explicit modules

IMPORT resolves a declared and exported symbol under a version constraint.



Kokum 0.29.0 core-language architecture

Figure 9. The application module imports exported functions from a separate math module.

16.1 The reusable module

Listing 20. src/math.kokum - exported reusable functions

VERIFIED

```

MODULE book.math VERSION "1.0.0"

EXPORT FUNCTION Add
EXPORT FUNCTION Percentage

FUNCTION Add(left AS DECIMAL, right AS DECIMAL) AS DECIMAL
  RETURN left + right
ENDFUNC

FUNCTION Percentage(amount AS DECIMAL, rate AS DECIMAL) AS DECIMAL
  RETURN amount * rate / 100M
ENDFUNC
  
```

Verification: Project check, module build, bundle, VM execution, and standalone execution verified

16.2 The importing application

Listing 21. src/app.kokum - required imports and a local alias

VERIFIED

```

MODULE book.application VERSION "1.0.0"

IMPORT FUNCTION Add FROM book.math VERSION "^1.0.0"
IMPORT FUNCTION Percentage AS Tax FROM book.math VERSION "^1.0.0"
  
```

```

EXPORT FUNCTION Main

FUNCTION Main AS INTEGER
  LOCAL subtotal AS DECIMAL
  LOCAL taxAmount AS DECIMAL

  subtotal = Add(1500M, 499.95M)
  taxAmount = Tax(subtotal, 18M)

  ? "Subtotal:", STR(subtotal, 12, 2)
  ? "Tax:", STR(taxAmount, 12, 2)
  ? "Total:", STR(subtotal + taxAmount, 12, 2)
  RETURN 0
ENDFUNC

```

Verification: Project check, module build, bundle, VM execution, and standalone execution verified

Observed bundle output

```

Subtotal:      1999.95
Tax:           359.99
Total:         2359.94

```

Module keyword	Purpose
MODULE <i>name</i> VERSION "x.y.z"	logical identity; must precede ordinary declarations
EXPORT FUNCTION <i>Name</i>	places the declaration in the public module surface
IMPORT FUNCTION <i>Name</i> FROM <i>module</i> VERSION <i>constraint</i>	required import
IMPORT FUNCTION <i>Name</i> AS <i>Alias</i> ...	local alias without changing the exporter
OPTIONAL import	link may proceed unresolved; invoking the missing symbol raises a catchable error
Import kinds	FUNCTION, PROCEDURE, CLASS, GLOBAL, NATIVE, BUILTIN

VERSIONING Publish small, intentional module APIs

Only exported declarations are visible to importers. Treat export names and signatures as a compatibility contract, and use semantic versions plus lockfiles to make resolution reviewable.

Chapter summary

- A separate function file becomes a module with an explicit name and version.
- Imports name both the symbol and its source module and can apply a local alias.
- Only declarations listed by **EXPORT** form the public API.

PRACTICE Try this

Create a `book.validation` module that exports `RequirePositive`. Import it into the application and use a caret version constraint.

CHAPTER 17

17. Project Manifests, Lockfiles, Bundles, and Standalone Builds

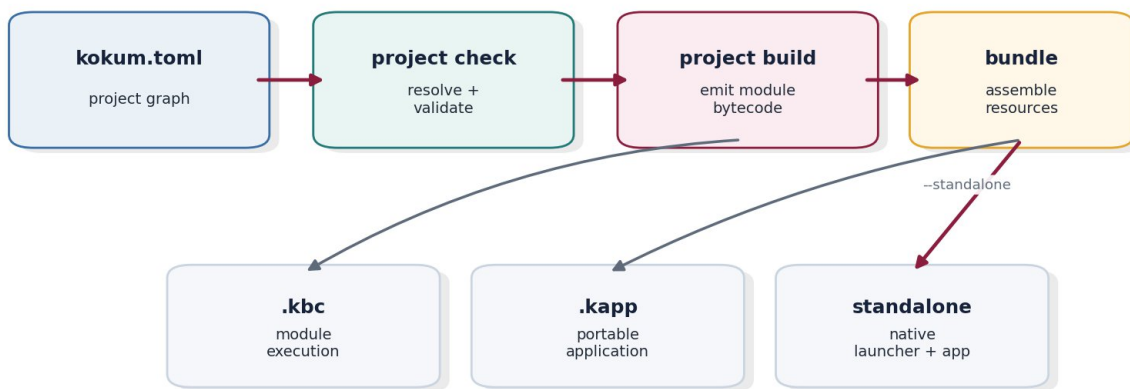
The project manifest is the reproducible description of an application graph. It connects module files, entry points, build directories, resources, and bundle targets.

IN THIS CHAPTER Learning outcomes

read a kokum.toml manifest; build and lock a module graph; create a portable .kapp bundle; create and test a standalone executable.

Project build and deployment artifacts

One source project can produce portable bytecode, a bundle, or a standalone executable.



Kokum 0.29.0 core-language architecture

Figure 10. Project validation leads to bytecode modules, an application bundle, or a standalone deliverable.

Listing 22. kokum.toml for the two-module application

VERIFIED

```

[project]
name = "book.module.example"
version = "1.0.0"
kind = "console"
entry_module = "book.application"
entry = "Main"

[build]
output_dir = "build"
lockfile = "kokum.lock"
cache_dir = ".kokum/cache"
write_lockfile = true

[bundle]
output = "dist/book-module-example.kapp"
target_os = "portable"
target_arch = "any"

[[module]]
name = "book.math"
version = "1.0.0"
source = "src/math.kokum"

[[module]]
name = "book.application"
version = "1.0.0"
source = "src/app.kokum"
  
```

Verification: Manifest parsed and project check/build/bundle/standalone paths passed

17.1 Project commands

Listing 23. Verified multi-file build sequence

VERIFIED

```
./kokumc project check kokum.toml
./kokumc project build kokum.toml
./kokumc project bundle kokum.toml
./kokumvm dist/book-module-example.kapp
./kokumc project build kokum.toml --standalone dist/book-module-example
./dist/book-module-example
```

Verification: All six stages passed in the final source tree

Manifest area	Key responsibility
[project]	name, version, project kind, entry module, entry routine
[build]	output directory, lockfile, cache, lockfile writing
[bundle]	output path and target OS/architecture
[[module]]	logical module name/version and source or package
[[resource]]	source file and target path inside the application

17.2 Bundle layout

A `.kapp` can contain `modules/*.kbc`, resources, resolved package payloads, and optional `native/<os>-<arch>` content. A bundle with no native payload and a portable target remains portable across compatible KokumVM hosts.

REPRODUCIBILITY

Commit the application lockfile

A lockfile records the versions and identities selected during resolution. Rebuild release artifacts from a clean checkout and verify that the lockfile and source manifest produce the same application graph.

Chapter summary

- `kokum.toml` is the authoritative project graph and build policy.
- `kokum.lock` records resolved dependency identity.
- A `.kapp` is a portable application bundle; a standalone build adds a native launch form.

PRACTICE

Try this

Add a read-only resource file to the manifest, bundle the project, and confirm that the resource appears under the intended target path.

CHAPTER 18

18. Text File Input and Output

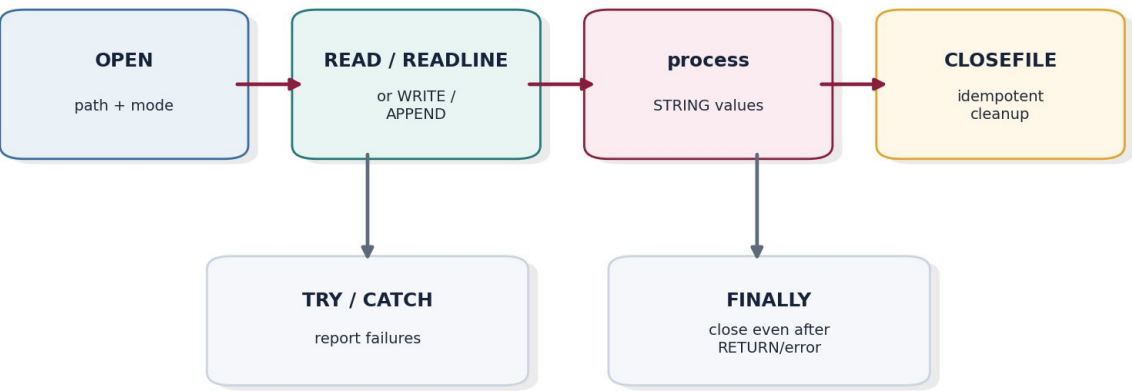
The built-in FILE API supports bounded UTF-8 text reads and writes. A file handle belongs to its runtime and should be closed in FINALLY.

IN THIS CHAPTER Learning outcomes

open a text file in the correct mode; read whole chunks or one line at a time; write and append text; close handles safely after errors.

Text file I/O lifecycle

FILE is an owner-runtime resource: open it, use it, and close it deterministically.



Text-only UTF-8 API; one transfer is bounded to 64 MiB

Figure 11. Open, use, process, and close a FILE value under structured error handling.

Mode	Meaning
r	read existing file
w	write with truncation/create
rw / r+	read/write existing file
w+	read/write with truncation/create
a	append/create
a+	read and append/create

Function	Behavior
OPEN(path, mode)	create an owner-runtime FILE handle
READ(file [, byteCount])	read UTF-8 text; returns remaining text or bounded bytes
READLINE(file)	read one line, strip LF/CRLF, return empty string for a blank line and NULL at EOF
WRITE(file, text)	write at current position
APPEND(file, text)	write at end
CLOSEFILE(file)	idempotently close the handle

Listing 24. Write and read a file line by line - 17_file_io.kokum

VERIFIED

```
FUNCTION Main AS INTEGER
  LOCAL file AS FILE
  LOCAL line
  LOCAL reading AS LOGICAL

  TRY
    file = OPEN("book-file-example.txt", "w")
    WRITE(file, "Alpha\nBeta\nGamma\n")
    CLOSEFILE(file)

    file = OPEN("book-file-example.txt", "r")
    reading = .T.
    DO WHILE reading
      line = READLINE(file)
      IF line = NULL
        reading = .F.
      ELSE
        ? line
      ENDIF
    ENDDO
  CATCH error
    ? "File error:", STR(error)
    RETURN 1
  FINALLY
    CLOSEFILE(file)
  ENDTRY

  RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

Alpha
Beta
Gamma

BOUNDARY Text API, not arbitrary binary storage

The initial FILE interface accepts UTF-8 text and rejects embedded NUL data. A single transfer is limited to 64 MiB. Stream large inputs with repeated **READ** or **READLINE** calls.

Chapter summary

- **READLINE** advances the file pointer and returns **NULL** only at EOF.
- **CLOSEFILE** is idempotent and belongs in **FINALLY**.
- FILE handles are runtime-owned and are not copied into async workers.

PRACTICE Try this

Modify the example to append a fourth line, reopen the file, and count nonblank lines without retaining the whole file.

CHAPTER 19

19. Runtime Memory and Garbage Collection

Most Kokum values are managed automatically. Explicit GC functions are diagnostic and operational controls, not substitutes for correct resource cleanup.

IN THIS CHAPTER Learning outcomes

distinguish managed values from native resources; understand cycle collection; inspect GC statistics; release resource handles deterministically.

Arrays, maps, strings, codeblocks, objects, and errors participate in managed memory. Cycles are collectible. Native resources such as FILE, DATABASE, WEBSOCKET, task, and synchronization objects also have runtime ownership rules; close or release them through their APIs rather than waiting for memory pressure.

Function	Use
<code>GCSTATS()</code>	return a MAP of runtime counters and collection statistics
<code>GCCOLLECT()</code>	request a collection and return the number of reclaimed cycle members
<code>GCDISABLE()</code>	temporarily disable automatic collection for a carefully bounded section
<code>GCENABLE()</code>	restore automatic collection

Listing 25. Create and collect a self-referential array - 21_runtime_memory.kokum

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL before AS MAP
  LOCAL after AS MAP
  LOCAL cycle AS ANY
  LOCAL collected AS INTEGER

  before = GCSTATS()
  cycle = []
  AADD(cycle, cycle)
  cycle = NULL

  collected = GCCOLLECT()
  after = GCSTATS()

  ? "Collected:", collected
  ? "Collections:", after.collections
  ? "Runtime:", before.runtimeId
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

Collected: 1
Collections: 1
Runtime: 1

```

IMPORTANT Memory ownership is not resource lifetime

Garbage collection reclaims unreachable managed values. It does not define when a transaction should roll back, a socket should close, a lock should release, or a file buffer should flush. Use explicit APIs and `FINALLY` for those invariants.

Chapter summary

- Managed cycles can be reclaimed by the runtime collector.
- `GCSTATS` and `GCCOLLECT` are useful for tests and diagnostics.
- Native resource cleanup remains an application responsibility.

PRACTICE Try this

Build a cycle containing a MAP and ARRAY, clear the last external reference, and compare GCSTATS before and after GCCOLLECT.

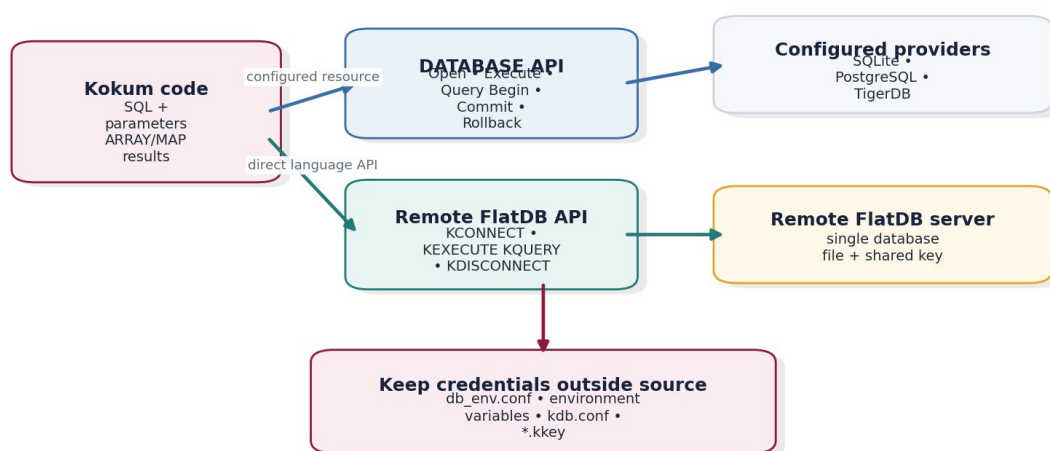
PART IV

Databases, Networking, and Concurrency

Database APIs, WebSocket clients, isolated tasks, named threads, and safe shared state.

Database paths in core Kokum

Use the provider-neutral DATABASE API for configured providers or KCONNECT for Remote FlatDB.



Kokum 0.29.0 core-language architecture

CHAPTER 20

20. Database Programming

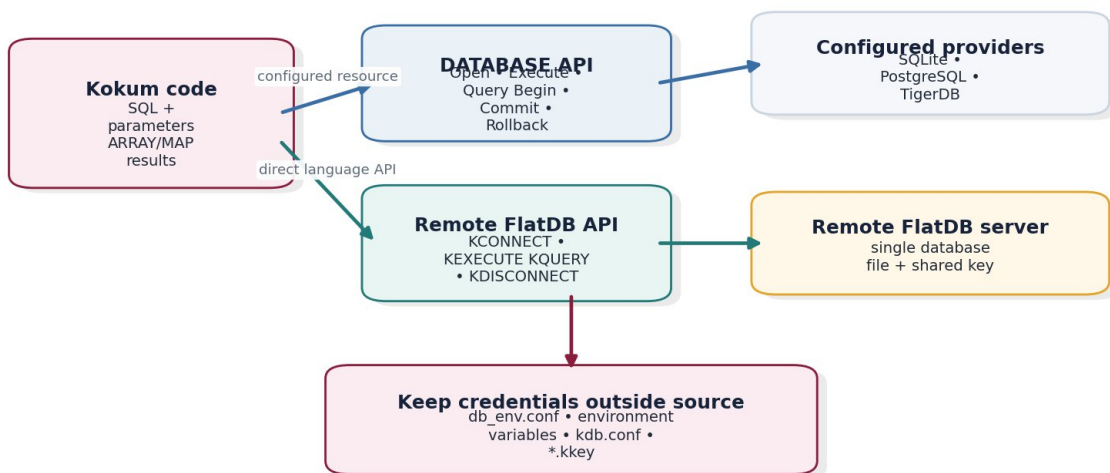
Kokum supports provider-neutral DATABASE resources and a direct Remote FlatDB language API. Keep connection policy and credentials outside source.

IN THIS CHAPTER Learning outcomes

use parameterized provider-neutral operations; structure transactions; connect to Remote FlatDB; handle database configuration securely.

Database paths in core Kokum

Use the provider-neutral DATABASE API for configured providers or KCONNECT for Remote FlatDB.



Kokum 0.29.0 core-language architecture

Figure 12. Core programs can use a configured DATABASE provider or the direct Remote FlatDB client API.

20.1 Provider-neutral DATABASE API

A configured **DATABASE** value exposes a common surface across SQLite, PostgreSQL, and TigerDB providers. The application receives a typed resource such as **PUBLIC dbMain AS DATABASE**; provider settings live in **db_env.conf** or another reviewed deployment configuration. Use **?** placeholders instead of concatenating untrusted values into SQL.

Listing 26. Provider-neutral transaction module - database_provider.kokum

VERIFIED

```

PUBLIC dbMain AS DATABASE

FUNCTION SaveCustomer(name AS STRING, city AS STRING) AS INTEGER
  LOCAL affected AS INTEGER

  dbMain.Begin()
  TRY
    affected = dbMain.Execute( ;
      "INSERT INTO customers(name, city) VALUES (?, ?)", ;
      [name, city])
    dbMain.Commit()
    RETURN affected
  CATCH error
    dbMain.Rollback()
    ? "Database error:", STR(error)
    RETURN 0
  ENDTRY
ENDFUNC

FUNCTION Main AS INTEGER
  && dbMain is supplied through the application's database configuration.
  ? "Provider-neutral database module compiled successfully"
  RETURN 0

```

ENDFUNC

Verification: Semantic check and portable bytecode compilation verified; live provider execution requires deployment configuration

Method	Purpose
<code>Open() / Close()</code>	manage the configured provider connection
<code>IsOpen()</code>	inspect connection state
<code>Execute(sql [, parameters])</code>	run DDL/DML and return provider-defined affected-row information
<code>Query(sql [, parameters])</code>	return an ARRAY of row MAPs
<code>QueryOne(...)</code>	return one row or NULL
<code>QueryValue(...)</code>	return one scalar
<code>QueryTable(...)</code>	table-oriented result form
<code>Begin() / Commit() / Rollback()</code>	explicit transaction control
<code>CreateTableFromJson(...)</code>	transactional JSON-driven table creation

20.2 Remote FlatDB - a fully runnable client

Remote FlatDB is a passive network database service backed by a single database file. Client and server share a binary key file. **KCONNECT USING** reads a **kdb.conf** beside the application, and **KQUERY** returns an ARRAY of row MAPs.

Listing 27. Authenticated Remote FlatDB DDL, DML, and queries - main.kokum

VERIFIED

```

FUNCTION Main AS INTEGER
  LOCAL rows AS ARRAY

  KCONNECT USING kdb.conf
  IF .NOT. KCONNECTED()
    ? "Connection failed"
    RETURN 1
  ENDIF

  KEXECUTE("CREATE TABLE customers (id INTEGER PRIMARY KEY, name TEXT, city TEXT)")
  KEXECUTE("INSERT INTO customers (id, name, city) VALUES (1, 'Anil', 'Ratnagiri')")
  KEXECUTE("INSERT INTO customers (id, name, city) VALUES (2, 'Meera', 'Pune')")

  rows = KQUERY("SELECT id, name, city FROM customers ORDER BY id")
  ? JSONENCODE(rows)

  KEXECUTE("UPDATE customers SET city='Mumbai' WHERE id=2")
  ? JSONENCODE(KQUERY("SELECT name, city FROM customers WHERE id=2"))

  KDISCONNECT()
  RETURN 0
ENDFUNC

```

Verification: Local server, source execution, bytecode execution, and output parity verified

Listing 28. kdb.conf used by the verified example

VERIFIED

```

[connection]
host = 127.0.0.1
port = 39237
database = company.kfdb
key_file = company.kkey
connect_timeout_ms = 2000
request_timeout_ms = 15000
max_message_bytes = 16777216

```

Verification: Configuration parsed by the local integration run; choose a deployment-specific port

Observed Remote FlatDB output

```

[{"id":1,"name":"Anil","city":"Ratnagiri"}, {"id":2,"name":"Meera","city":"Pune"}]
[{"name":"Meera","city":"Mumbai"}]

```

KEY MANAGEMENT Generate a real binary key

For a new environment, create a 64-byte key with `openssl rand -out company.kkey 64`, copy the exact

protected file to both endpoints, and never print or commit its bytes.

Chapter summary

- The provider-neutral API uses the same DATABASE methods across supported providers.
- Parameters belong in an ARRAY passed separately from SQL.
- Remote FlatDB uses **KCONNECT**, **KEXECUTE**, **KQUERY**, and **KDISCONNECT** with a shared key file.

PRACTICE Try this

Add a transaction that inserts two rows through a configured DATABASE provider. Then rewrite the same logical operation for Remote FlatDB and compare configuration and result shapes.

CHAPTER 21

21. WebSocket Client Programming

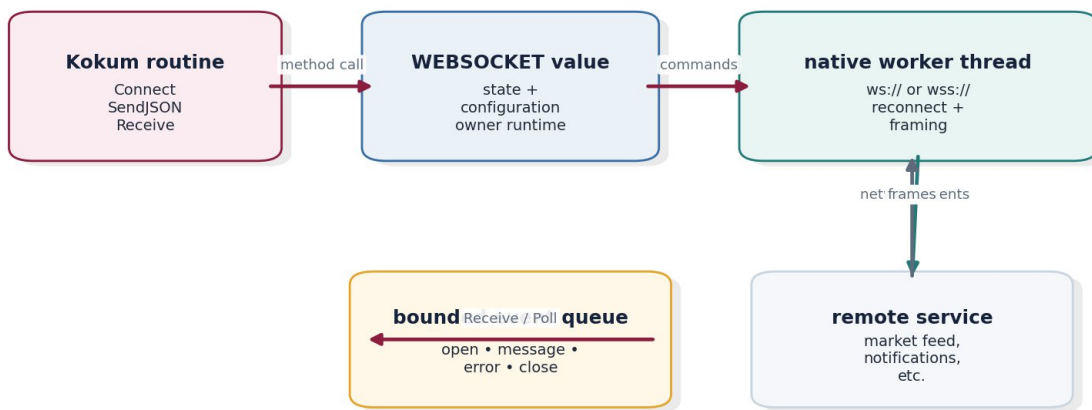
The nonvisual WebSocket client is an owner-runtime resource with connection configuration, authentication helpers, a native worker, reconnect policy, and a bounded event queue.

IN THIS CHAPTER Learning outcomes

configure a WEBSOCKET resource; connect and send structured messages; receive queued events with timeouts; close cleanly and reason about ownership.

WebSocket client execution model

A configured WEBSOCKET resource owns the connection worker and a bounded event queue.



Client-side WebSocket support; the HTTP service host is a separate feature

Figure 13. A WEBSOCKET value coordinates Kokum routines, a native network worker, and a bounded event queue.

Core source declares a typed **PUBLIC** WebSocket value. Deployment/application configuration supplies the resource instance; the book does not cover visual designer steps. This boundary keeps network settings and credentials outside reusable business routines.

Listing 29. Configured secure WebSocket client - websocket_client.kokum

VERIFIED

```

PUBLIC wsFeed AS WEBSOCKET

FUNCTION ConfigureFeed AS INTEGER
  LOCAL event AS MAP

  wsFeed.SetUrl("wss://feed.example.com/stream")
  wsFeed.SetBearerToken("access-token")
  wsFeed.SetQueryParam("client", "kokum")
  wsFeed.SetReconnect(.T., 1000, 10)

  wsFeed.Connect()
  wsFeed.SendJSON({"action": "subscribe", "symbols": ["NIFTY", "BANKNIFTY"]})

  event = wsFeed.Receive(500)
  IF event != NULL
    ? JSONENCODE(event)
  ENDIF

  wsFeed.Close(1000, "Normal shutdown", 3000)
  RETURN 0
ENDFUNC

FUNCTION Main AS INTEGER
  && wsFeed is supplied by a configured WebSocket application component.
  ? "WebSocket client module compiled successfully"
  RETURN 0

```

ENDFUNC

Verification: Semantic check and portable bytecode compilation passed; official local ws:// and wss:// loopback integration suite passed

API area	Representative methods or properties
Endpoint and protocol	<code>SetUrl</code> , <code>SetProtocols</code> / <code>SetSubprotocols</code>
Authentication	<code>SetHeader</code> , <code>SetBearerToken</code> , API/feed key helpers, Basic authentication, cookie and query parameters
Reconnect	<code>SetReconnect(enabled, delayMs, attempts)</code>
Lifecycle	<code>Connect</code> , <code>Close</code> , <code>Disconnect</code> , state properties
Sending	<code>SendText</code> , <code>SendJSON</code> , <code>SendBinary</code> , <code>Ping</code>
Receiving	<code>Wait</code> , <code>Receive</code> , <code>Poll</code> , <code>ReceiveText</code> , <code>ReceiveJSON</code> , <code>Drain</code> , <code>ClearMessages</code> , <code>Snapshot</code>

21.1 Event-driven receive loop

Use a finite receive timeout so the routine can check shutdown state, reconnect state, or service health. Treat `open`, `message`, `error`, and `close` events as explicit states. Validate every decoded message before applying it to orders, database rows, or shared state.

OWNERSHIP Do not pass WEBSOCKET into an async task

The resource owns a connection worker and queue tied to its runtime. Keep socket I/O in the owner session and pass ordinary decoded MAP/ARRAY data to workers or channels.

Chapter summary

- The WebSocket client supports `ws://` and `wss://`, authentication configuration, reconnect, sending, and bounded receive queues.
- The resource is supplied by application configuration and remains owner-runtime state.
- Network data must be validated before it changes durable or trading state.

PRACTICE Try this

Write a receive loop that waits 500 ms, handles a NULL timeout, processes only `message` events, and exits cleanly on `close`.

CHAPTER 22

22. Async Functions and Tasks

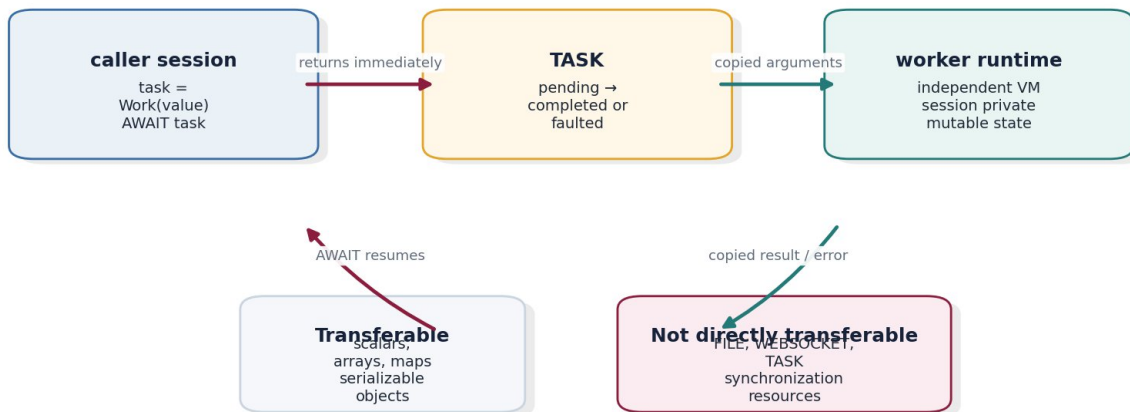
An ASYNC routine returns a TASK immediately. The dedicated KokumVM runs the task in an isolated worker runtime and copies transferable arguments and results across the boundary.

IN THIS CHAPTER Learning outcomes

declare and call ASYNC routines; await a task result; inspect task state; respect value-transfer restrictions.

Async tasks isolate runtime state

Arguments and results cross the boundary by value; the dedicated VM uses OS worker threads.



Kokum 0.29.0 core-language architecture

Figure 14. Async tasks use isolated runtime state and copy values across the task boundary.

Listing 30. Start and await an asynchronous calculation - 18_async_tasks.kokum

VERIFIED

```

ASYNC FUNCTION SumLater(values AS ARRAY) AS INTEGER
  LOCAL total AS INTEGER
  LOCAL value AS INTEGER

  SLEEP(50)
  total = 0
  FOR EACH value IN values
    total = total + value
  NEXT
  RETURN total
ENDFUNC

FUNCTION Main AS INTEGER
  LOCAL task AS TASK
  LOCAL result AS INTEGER

  task = SumLater([10, 20, 30])
  ? "Initial status:", task.Status
  result = AWAIT task
  ? "Result:", result
  ? "Completed:", task.IsCompleted
  RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

```

Initial status: completed
Result: 60
Completed: .T.

```

Task operation	Meaning
<code>AWAIT task</code>	block the current session until result or error
<code>task.Await(...)</code> / <code>task.Result(...)</code>	method equivalents, optionally with timeout where supported
<code>task.Wait(...)</code>	wait for completion without emphasizing the result
State properties	Status, completion, success/fault state, result/error information
Transfer model	arguments, receiver state, globals, and result cross by value
Not transferable	cyclic mutable graphs, CODEBLOCK, TASK, FILE, WEBSOCKET, unknown native resources and synchronization resources
DATABASE transfer	provider configuration is copied and the worker opens an independent connection

EXECUTION PARITY Reference runners may complete synchronously

The AST and canonical-IR reference executors preserve observable task results but may return an already completed task. Use the dedicated KokumVM when testing real operating-system concurrency and timing.

Chapter summary

- `ASYNC FUNCTION`, `ASYNC PROCEDURE`, and `ASYNC METHOD` return TASK values.
- `Main` is not async; it starts and awaits tasks explicitly.
- Mutable state is isolated and transferable data is copied.

PRACTICE Try this

Start two independent async calculations before awaiting either one. Await both and combine their results, then explain why the inputs cannot be shared mutable arrays.

CHAPTER 23

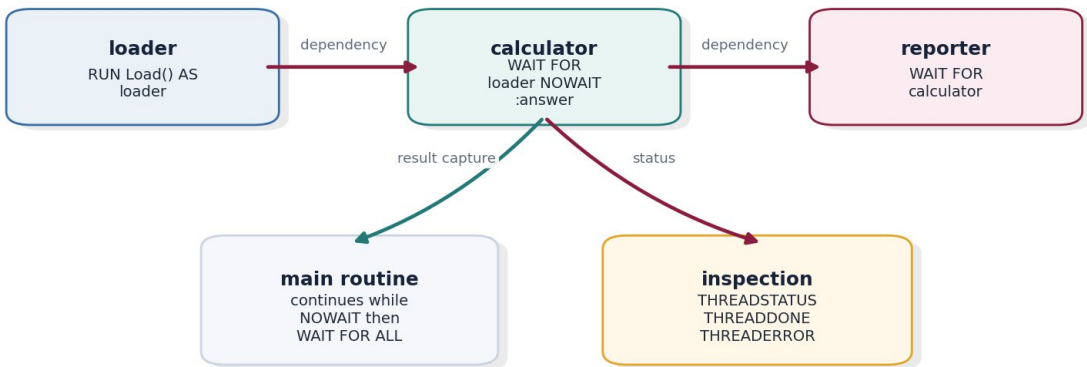
23. Named Threads, Dependencies, and Worker Pool

Named thread syntax makes long-running jobs visible. A RUN statement can capture a result, start without waiting, depend on another job, and be inspected later.

IN THIS CHAPTER Learning outcomes
start named thread jobs; use NOWAIT and result capture; wait for one, several, or all jobs; inspect status and errors.

Named thread dependencies

RUN creates a named job; WAIT FOR establishes completion dependencies and result capture.



Kokum 0.29.0 core-language architecture

Figure 15. Named jobs can form an explicit dependency graph while Main continues.

Listing 31. Run a named calculation and capture its result - 19_named_threads.kokum

VERIFIED

```
FUNCTION Calculate(left AS INTEGER, right AS INTEGER) AS INTEGER
    SLEEP(40)
    RETURN left + right
ENDFUNC

FUNCTION Main AS INTEGER
    LOCAL answer AS INTEGER

    RUN Calculate(20, 22) AS calculation NOWAIT :answer
    ? "Main continues"
    WAIT FOR calculation
    ? "State:", THREADSTATUS(calculation)
    ? "Answer:", answer
    RETURN 0
ENDFUNC
```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

Main continues
State: COMPLETED
Answer: 42

Construct	Purpose
<code>RUN Work() AS job</code>	start a named job under the default wait policy

Construct	Purpose
<code>NOWAIT</code>	allow the caller to continue immediately
<code>:resultVariable</code>	copy the completed return value into a declared destination
<code>WAIT FOR job</code>	wait for one job
<code>WAIT FOR a, b</code>	wait for several named jobs
<code>WAIT FOR ALL</code>	join all jobs visible to the current thread scope
<code>TIMEOUT milliseconds</code>	bound a wait
Inspection	<code>THREADSTATUS</code> , <code>THREADDONE</code> , <code>THREADERROR</code> , <code>THREADID</code> , <code>THREADPOOLSIZE</code>

23.1 Dependency-aware RUN

Listing 32. A readable three-stage dependency chain

VERIFIED

```

RUN LoadData() AS loader NOWAIT
RUN Calculate() AS calculator WAIT FOR loader NOWAIT :answer
RUN Publish(answer) AS publisher WAIT FOR calculator NOWAIT
WAIT FOR publisher TIMEOUT 5000

```

Verification: Syntax and behavior are covered by the named-thread regression suite; adapt declarations to the application

FAILURE POLICY Inspect a dependency before trusting its result

A completed job can be successful or faulted. Check status/error when failure must select a fallback, retry, or compensating action. Do not assume that `WAIT FOR` converts every worker failure into a valid result.

Chapter summary

- Named threads expose job identity and result capture in source.
- `WAIT FOR` expresses joins and dependencies, with optional timeout.
- Thread inspection functions support diagnostics and failure policy.

PRACTICE Try this

Create a loader, transformer, and writer dependency chain. Add a finite timeout and define what the program should do when the transformer faults.

CHAPTER 24

24. Safe Shared State: Mutexes, Atomics, and Channels

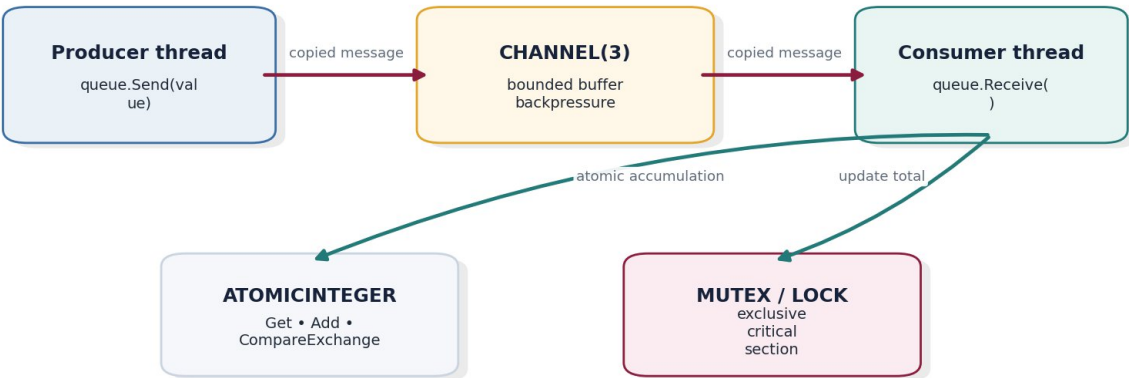
Concurrency is safest when jobs exchange copied messages. When state must be shared, use the smallest synchronization primitive that preserves the invariant.

IN THIS CHAPTER Learning outcomes

protect a critical section with a mutex; update counters atomically; move messages through a bounded channel; close a producer-consumer pipeline.

Safe shared state

Channels transfer copied messages; atomics and mutexes protect intentionally shared state.



Kokum 0.29.0 core-language architecture

Figure 16. Channels provide bounded message flow; atomics and mutexes protect deliberate shared state.

Primitive	Primary operations	Use
MUTEX()	Lock, TryLock, Unlock; LOCK ... ENDLOCK	multi-step critical section
ATOMICINTEGER(initial)	Get, Set, Exchange, Add, Increment, Decrement, CompareExchange	small lock-free integer state
CHANNEL(capacity)	Send, Receive, TrySend, TryReceive, Close	bounded producer-consumer messages

Listing 33. Bounded channel with atomic accumulation - 20_safe_shared_state.kokum (continued 1/2)

VERIFIED

```
FUNCTION Producer(queue AS CHANNEL)
    LOCAL i AS INTEGER

    FOR i = 1 TO 5
        queue.Send(i)
    NEXT
    queue.Close()
    RETURN 0
ENDFUNC

FUNCTION Consumer(queue AS CHANNEL, total AS ATOMICINTEGER)
    LOCAL value
    LOCAL receiving AS LOGICAL

    receiving = .T.
    DO WHILE receiving
        value = queue.Receive()
        IF value = NULL
            receiving = .F.
        ELSE
            total.Add(value)
        ENDIF
    ENDWHILE
ENDFUNC
```

```

        ELSE
            total.Add(value)
        ENDIF
    ENDDO
    RETURN 0
ENDFUNC

FUNCTION Main AS INTEGER
    LOCAL queue AS CHANNEL
    LOCAL total AS ATOMICINTEGER

```

Listing 33. Bounded channel with atomic accumulation - 20_safe_shared_state.kokum (continued 2/2)

VERIFIED

```

    queue = CHANNEL(3)
    total = ATOMICINTEGER(0)

    RUN Producer(queue) AS producer NOWAIT
    RUN Consumer(queue, total) AS consumer NOWAIT
    WAIT FOR producer, consumer

    ? "Total:", total.Get()
    RETURN 0
ENDFUNC

```

Verification: Source check, interpreter, bytecode, and KokumVM output verified

Observed output

Total: 15

24.1 Message and closure rules

Channel messages are copied across runtime boundaries. Native handles, tasks, codeblocks, cyclic mutable graphs, and synchronization values cannot be sent as ordinary messages. Close the channel when producers are finished so consumers can observe **NULL** and terminate.

DEADLOCK PREVENTION Define lock ordering and timeout policy

When more than one mutex is unavoidable, document one global acquisition order. Keep critical sections short, never perform unbounded network I/O while holding a lock, and use finite waits where a failure policy exists.

Chapter summary

- Prefer channels and copied messages over broad shared mutable state.
- Use ATOMICINTEGER for simple integer invariants and MUTEX for compound invariants.
- Closing a channel is part of the pipeline protocol.

PRACTICE Try this

Extend the producer-consumer program to use two producers. Explain why closing the channel requires coordination so one producer cannot close it while another is still sending.

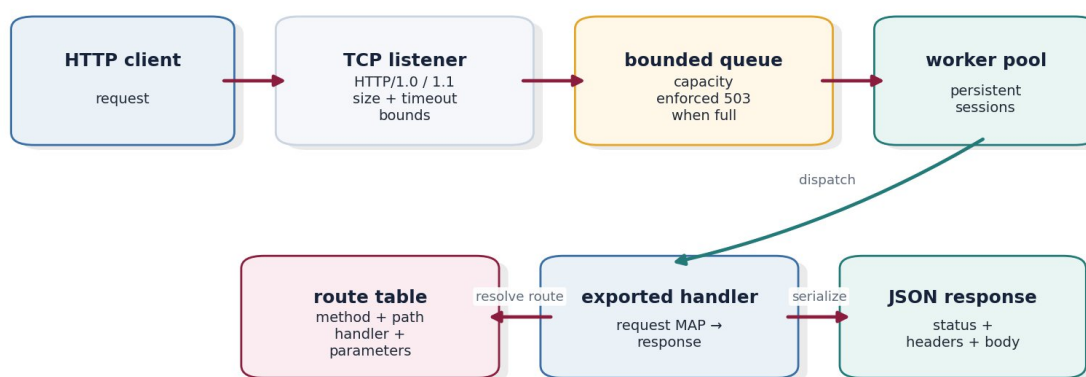
PART V

Backend Services and Deployment

Build bounded HTTP services, configure routes, test endpoints, and deploy behind a production boundary.

KokumVM HTTP microservice architecture

A fixed worker pool owns persistent VM sessions behind a bounded accepted-request queue.



Kokum 0.29.0 core-language architecture

CHAPTER 25

25. Writing HTTP Microservices

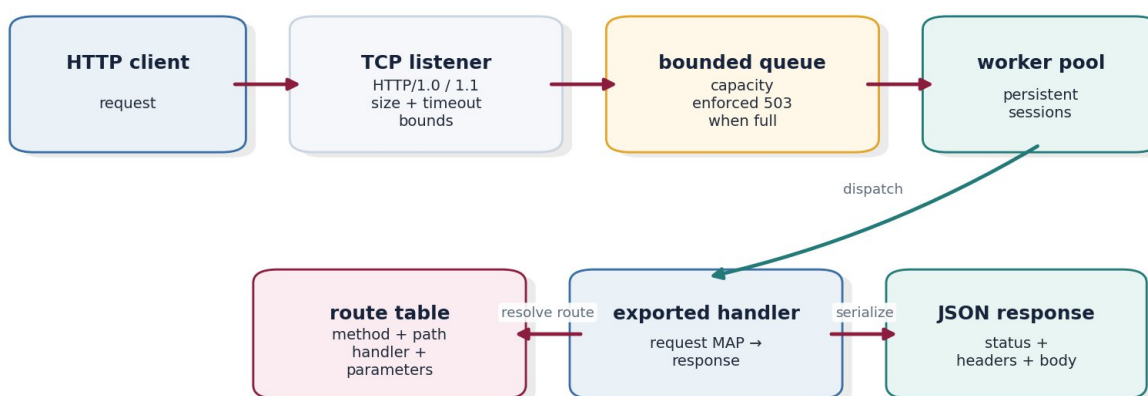
KokumVM can host a compiled module or application bundle as a bounded HTTP/1.1 service. Exported handlers receive a request MAP and return text, JSON-safe data, or a response MAP.

IN THIS CHAPTER Learning outcomes

write exported route handlers; read query, JSON, and path-parameter data; return structured HTTP responses; run the service with a fixed worker pool.

KokumVM HTTP microservice architecture

A fixed worker pool owns persistent VM sessions behind a bounded accepted-request queue.



Kokum 0.29.0 core-language architecture

Figure 17. A listener, bounded queue, and persistent VM worker pool execute exported Kokum handlers.

25.1 Complete service module

Listing 34. Three exported HTTP route handlers - service.kokum (continued 1/2)

VERIFIED

```

MODULE book.microservice VERSION "1.0.0"

EXPORT FUNCTION Main
EXPORT FUNCTION Hello
EXPORT FUNCTION Add
EXPORT FUNCTION Customer

FUNCTION Main AS INTEGER
  RETURN 0
ENDFUNC

FUNCTION Hello(request AS MAP) AS MAP
  LOCAL name AS STRING

  name = request["Query"]["name"]
  IF name = NULL
    name = "reader"
  ENDIF

  RETURN { ;
    "Status": 200, ;
    "Headers": {"X-Application": "Kokum Core Book"}, ;
    "Json": {"message": "Hello " + name, "requestId": request["RequestId"]} ;
  }
ENDFUNC

FUNCTION Add(request AS MAP) AS MAP

```

```
LOCAL body AS MAP
LOCAL result AS NUMBER
```

Listing 34. Three exported HTTP route handlers - service.kokum (continued 2/2)

VERIFIED

```
body = request["Json"]
result = body["left"] + body["right"]
RETURN {"Status": 200, "Json": {"result": result}}
ENDFUNC

FUNCTION Customer(request AS MAP) AS MAP
RETURN {"Json": {"id": request["Params"]["id"], "active": .T.}}
ENDFUNC
```

Verification: Project check, bundle build, live HTTP requests, and graceful stop verified

25.2 Server and route configuration

Listing 35. server.conf for the verified service (continued 1/2)

VERIFIED

```
[server]
listen = 127.0.0.1
port = 0
workers = 2
queue_capacity = 16
backlog = 32
max_request_bytes = 1048576
max_body_bytes = 262144
read_timeout_ms = 3000
write_timeout_ms = 3000
graceful_shutdown_ms = 5000
health_path = /health
server_name = Kokum-Core-Book
trace = false

[route hello]
method = GET
path = /hello
handler = Hello
parameters = 1

[route add]
method = POST
path = /add
handler = Add
parameters = 1

[route customer]
method = GET
path = /customers/:id
```

Listing 35. server.conf for the verified service (continued 2/2)

VERIFIED

```
handler = Customer
parameters = 1
```

Verification: Service bound an ephemeral local port and served all configured routes

Observed local HTTP requests

```
health 200 {"status":"ok","workers":2,"completed":0,"request_id":"1788663483253-1"}
hello 200 {"message":"Hello Anil","requestId":"1788663483254-2"}
add 200 {"result":42}
customer 200 {"id":"101","active":true}
```

25.3 Request and response model

Request field	Contents
Method	HTTP method
Path	decoded path without query string
Target	original request target
QueryString	original query string without ?

Request field	Contents
HttpVersion	HTTP/1.0 or HTTP/1.1
RemoteAddress / RemotePort	numeric client endpoint
RequestId	server-generated identifier
Body	request body STRING
Headers	MAP with lower-case header names
Query	URL-decoded query MAP
Params	URL-decoded route-parameter MAP
Json	decoded JSON or NULL

Handler return	HTTP result
NULL	empty 200 response
STRING	text/plain UTF-8 response
JSON-safe scalar/collection	JSON response
Response MAP	Status, Headers, ContentType, Body, or Json; Json takes precedence over Body

CAPACITY The queue is deliberately bounded

When the accepted-request queue is full, the service returns 503 rather than creating unbounded threads or silently dropping work. Choose workers, queue capacity, request limits, and timeouts from measured workload behavior.

Chapter summary

- Handlers must be exported because the host resolves them through the module callback API.
- One-parameter handlers receive a structured request MAP.
- The service uses fixed workers and a bounded queue with explicit 503 behavior.

PRACTICE Try this

Add a `POST /customers` handler that requires a JSON `name`, returns status 201, and adds a Location header. Test malformed JSON and a missing field.

CHAPTER 26

26. Deployment, Configuration, Security, and Testing

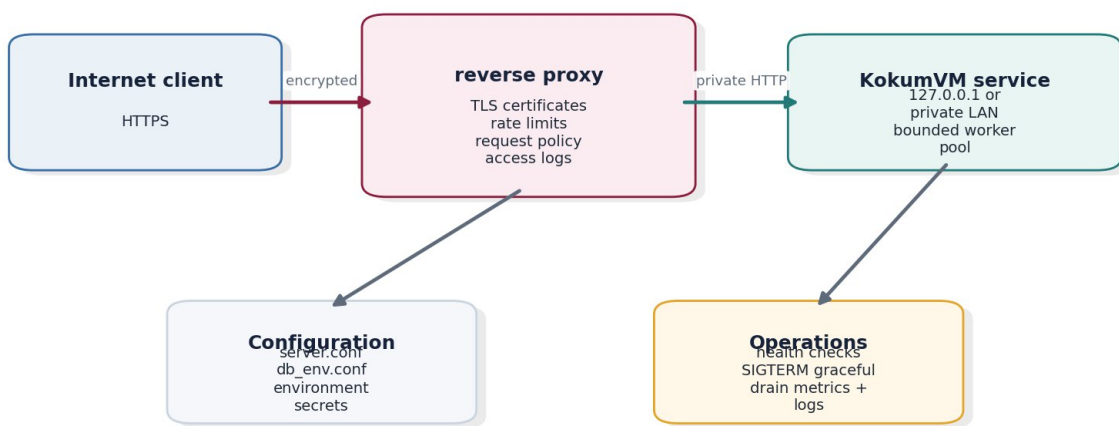
A correct handler is only one layer of a service. Production operation requires explicit configuration, private bindings, TLS termination, health checks, bounded resources, logs, and graceful shutdown.

IN THIS CHAPTER Learning outcomes

package and start a service; use configuration and ready files; place a reverse proxy at the public boundary; test health, limits, failures, and shutdown.

Production deployment boundary

Keep the Kokum service private; terminate public TLS and policy at a reviewed reverse proxy.



Kokum 0.29.0 core-language architecture

Figure 18. Terminate public TLS and policy at a reviewed reverse proxy while KokumVM listens privately.

26.1 Build and start

Listing 36. Verified service build and startup sequence

VERIFIED

```

./kokumc project check kokum.toml
./kokumc project bundle kokum.toml
./kokumvm serve dist/book-microservice.kapp \
  --config server.conf \
  --ready-file service.ready
  
```

Verification: Bundle started, wrote the ready URL, served four requests, and stopped gracefully

26.2 Public deployment boundary

The Phase 4.15 service foundation supports bounded HTTP/1.0 and HTTP/1.1, exact and `:parameter` routes, Content-Length bodies, JSON decoding, health checks, one request per connection, structured errors, and graceful shutdown. It does not provide native public TLS, HTTP/2, chunked request bodies, WebSocket upgrade, multipart upload, middleware, static-file serving, or streaming responses.

- Bind KokumVM to `127.0.0.1` or a private service network.
- Terminate TLS, public rate limits, request filtering, and external access logs at Nginx, Apache, Caddy, HAProxy, or another reviewed reverse proxy.
- Keep database passwords, bearer tokens, client private keys, and FlatDB keys out of source and ordinary logs.
- Set finite body, request, queue, read, write, and graceful-shutdown bounds.
- Use the health endpoint for readiness/liveness policy appropriate to the deployment.
- On SIGINT or SIGTERM, allow active and queued requests to drain within the configured deadline.

26.3 Verification matrix for a service release

Test	Expected result
Semantic/project check	no errors and no unexpected warnings
Bundle execution	entry module and all route handlers resolve
Health request	200 response with service status
Happy-path route tests	status, headers, JSON shape, and values match contract
Malformed JSON	400 before handler execution
Unknown route / method	structured 404/405 behavior as configured
Oversized request	bounded rejection without memory growth
Queue saturation	503 response rather than unbounded thread creation
SIGTERM	listener closes, work drains, shutdown handlers run, process exits
Reverse-proxy test	correct forwarded policy, timeout behavior, TLS, and body limits

OPERATIONAL RULE Configuration changes are code changes

Review, test, version, and roll back service configuration with the same discipline as source. A timeout, worker count, route, TLS policy, or credential path can change system behavior as significantly as a function edit.

Chapter summary

- KokumVM services should remain behind a private deployment boundary for public workloads.
- The service host is bounded by worker, queue, request, body, and timeout settings.
- Release testing must include invalid input, overload, and graceful shutdown, not only happy paths.

PRACTICE Try this

Deploy the verified service behind a local reverse proxy, enable TLS with a development certificate, and test body-size, timeout, and graceful-restart behavior.

APPENDIX A

Core Syntax Quick Reference

Routine and class forms

Listing 37. Declaration templates

VERIFIED

```

FUNCTION Name(arguments) AS Type
    RETURN value
ENDFUNC

PROCEDURE Name(arguments)
    && statements
ENDPROC

ASYNC FUNCTION Name(arguments) AS Type
    RETURN value
ENDFUNC

CLASS Name [AS BaseClass]
    PROPERTY Value AS Type = default
    METHOD Init(arguments)
        THIS.Value = value
    ENDMETHOD
ENDCLASS

```

Verification: Templates reflect the implemented grammar; replace placeholders with valid declarations

Control-flow forms

Listing 38. Decision and loop templates

VERIFIED

```

IF condition
    statements
ELSEIF otherCondition
    statements
ELSE
    statements
ENDIF

DO CASE
CASE condition
    statements
OTHERWISE
    statements
ENDCASE

FOR i = 1 TO 10 STEP 2
    statements
NEXT

DO WHILE condition
    statements
ENDDO

FOR EACH key, value IN collection
    statements
NEXT

```

Verification: Templates reflect the implemented grammar

Error, task, and thread forms

Listing 39. Error and concurrency templates

VERIFIED

```

TRY
    statements
CATCH error
    recovery
FINALLY
    cleanup
ENDTRY

task = AsyncWork(value)
result = AWAIT task

RUN Work(value) AS worker NOWAIT :result

```

```
WAIT FOR worker TIMEOUT 5000
```

Verification: Templates reflect verified language features

Module forms

Listing 40. Module declaration template	VERIFIED
<pre>MODULE company.module VERSION "1.0.0" IMPORT FUNCTION Parse FROM company.text VERSION "^1.0.0" IMPORT FUNCTION Format AS Render FROM company.text VERSION "^1.0.0" EXPORT FUNCTION Main</pre>	

Verification: Module syntax verified by the two-module project

APPENDIX B

Core Built-in Function Quick Reference

This table focuses on nonvisual core functions. Provider objects and specialized APIs may expose additional methods.

Area	Functions or constructors
Text and conversion	STR, LEN, UPPER, LOWER, VAL, DECIMAL, TYPEOF
Date/time	DATE, DATETIME
Numeric	ABS, INT, MIN, MAX; scalar mathematics and statistics functions documented by their focused references
JSON	JSONENCODE, JSONDECODE
Arrays	AADD, ASET, AINS, ADEL, LEN
Maps	MAPSET, HASKEY, MAPREMOVE, KEYS, VALUES, LEN
Codeblocks	EVAL
Objects/reflection	NEW syntax, CLASSNAME, PROPERTIES, METHODS, HASPROPERTY, HASMETHOD, GETPROPERTY, SETPROPERTY
File I/O	OPEN, READ, READLINE, WRITE, APPEND, CLOSEFILE
Remote FlatDB	KCONNECT, KCONNECTED, KEXECUTE, KQUERY, KDISCONNECT
Tasks/time	AWAIT syntax, TASK methods, SLEEP
Named threads	THREADSTATUS, THREADDONE, THREADERROR, THREADID, THREADPOOLSIZE
Synchronization	MUTEX, LOCK/UNLOCK, ATOMICINTEGER, CHANNEL
Memory	GCSTATS, GCCOLLECT, GCDISABLE, GCENABLE
Pattern matching	REGEXCOMPILE, REGEXMATCH, REGEXFIND, REGEXFINDALL, REGEXREPLACE, REGEXSPLIT, REGEXESCAPE - see the separate RegEx manual

APPENDIX C

Files, Commands, and Artifact Reference

Compiler and VM commands

Command	Purpose
<code>kokumc check source.kokum</code>	semantic validation without execution
<code>kokumc run source.kokum</code>	source/reference execution
<code>kokumc run-ir source.kokum</code>	canonical IR execution
<code>kokumc run-bytecode source.kokum</code>	reference bytecode execution
<code>kokumc compile source.kokum -o program.kbc</code>	emit portable bytecode
<code>kokumvm program.kbc</code>	execute bytecode in the dedicated VM
<code>kokumc project check kokum.toml</code>	validate and resolve project graph
<code>kokumc project build kokum.toml</code>	build project modules
<code>kokumc project bundle kokum.toml</code>	create <code>.kapp</code>
<code>kokumc project build kokum.toml --standalone output</code>	create standalone application form
<code>kokumvm application.kapp</code>	run a bundle
<code>kokumvm serve application.kapp --config server.conf</code>	host an HTTP service

File checklist

File	Release check
<code>.kokum</code>	source reviewed and all exported contracts intentional
<code>kokum.toml</code>	entry, modules, resources, and targets correct
<code>kokum.lock</code>	resolved identities reviewed and reproducible
<code>.kbc</code>	generated from clean source; VM version compatible
<code>.kapp</code>	bundle contents and target policy audited
<code>db_env.conf</code>	not committed with production secrets
<code>kdb.conf</code> / <code>.kkey</code>	endpoint correct; key protected and identical at both ends
<code>server.conf</code>	private listen policy, routes, workers, bounds, timeouts, and health path reviewed

APPENDIX D

Verification Matrix and Troubleshooting

Example verification matrix

Example group	Semantic check	Source run	Bytecode	External VM
Chapters 1-15 standalone examples	PASS	PASS	PASS	output matched
Chapter 18 file I/O	PASS	PASS	PASS	output matched
Chapters 22-24 concurrency	PASS	PASS	PASS	stable results matched; initial task state may differ
Chapter 19 GC example	PASS	PASS	PASS	output matched
Two-module project	project PASS	bundle and standalone PASS	module <code>.kbc</code> emitted	bundle/standalone output matched
Remote FlatDB	PASS	live local server PASS	PASS	query output matched
WebSocket client	PASS	official ws/wss loopback PASS	PASS	integration suite PASS
HTTP microservice	project PASS	live endpoint tests PASS	bundle emitted	service host PASS

Common diagnostic patterns

Symptom	Likely cause	First action
undeclared name	assignment or use before declaration; spelling mismatch	add or correct the typed declaration
type mismatch	incompatible assignment or DECIMAL/NUMBER mixing	validate input and convert explicitly
invalid array index	zero, negative, noninteger, or out of range	check <code>1..LEN(array)</code>
missing map data	key absent and read returned NULL	use <code>HASKEY</code> when absence matters
file operation failure	invalid mode/path/permissions or closed handle	catch the error and close in <code>FINALLY</code>
module import unresolved	symbol not exported, wrong module/version, stale lockfile	inspect export, constraint, manifest, and lock resolution
task transfer failure	argument/result contains a nontransferable resource or cycle	send plain copied data and reopen resources inside worker
thread wait timeout	job blocked, slow, or dependency faulted	inspect <code>THREADSTATUS</code> and <code>THREADERROR</code>
database failure	connection policy, parameter mismatch, transaction state, privilege, or SQL error	log safe context, rollback, verify provider config
service 503	bounded queue is full	measure saturation; tune capacity/workers or reduce handler latency
WebSocket no event	timeout, connection state, queue policy, or remote silence	inspect state/snapshot and use finite receive waits

A disciplined defect report

1. Record the exact Kokum version, platform, compiler/VM command, and build configuration.
2. Reduce the failure to the smallest complete source or project that still reproduces it.
3. Include semantic diagnostics, stdout/stderr, configuration with secrets removed, and expected versus observed behavior.

- 4.State whether source execution, bytecode-reference execution, and external KokumVM agree.
- 5.For concurrency or network issues, include timestamps, finite timeouts, thread/task states, and a bounded log excerpt.
- 6.For services, include the request method/path/headers/body size, response, health status, and shutdown behavior.

FINAL NOTE Use the separate pattern manual for RegEx and Natural Patterns

This book intentionally keeps pattern matching out of the core-language chapters. The dedicated Kokum Regular Expressions and Natural Patterns Manual contains the complete RegEx API, Natural FIND/REPLACE/SPLIT/VALIDATE/ITERATE syntax, captures, diagnostics, and Pattern Inspector workflow.

Implementation references used for this edition

The book was derived from the latest executable source tree at the writing of this book, focused language contracts, project/module specifications, provider documentation, concurrency specifications, service-host documentation, and the verified example suite. Source behavior takes precedence over historical notes when a document describes an earlier phase.

End of Kokum Core Language Programming