

Package ‘arl’

May 7, 2026

Type Package

Title Embedded Lisp Dialect

Version 0.1.4

Description Provides a Scheme-inspired Lisp dialect embedded in R, with macros, tail-call optimization, and seamless interoperability with R functions and data structures. (The name ‘arl’ is short for ‘An R Lisp.’) Implemented in pure R with no compiled code.

License MIT + file LICENSE

Depends R (>= 4.0)

Imports R6 (>= 2.5.0)

Suggests testthat (>= 3.0.0), covr (>= 3.6.0), lintr (>= 3.3.0), withr (>= 2.5.0), profvis (>= 0.3.7), bench (>= 1.1.2), jsonlite (>= 2.0.0), knitr (>= 1.51), rmarkdown (>= 2.30), jinjar (>= 0.3.0)

URL <https://github.com/wwbrannon/arl>,
<https://doi.org/10.5281/zenodo.18740487>

BugReports <https://github.com/wwbrannon/arl/issues>

Language en-US

Encoding UTF-8

ByteCompile true

NeedsCompilation no

RoxygenNote 7.3.3

Config/testthat/edition 3

VignetteBuilder knitr

Author William Brannon [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-1435-8535>)

Maintainer William Brannon <will.brannon+cran@gmail.com>

Repository CRAN

Date/Publication 2026-03-19 14:00:09 UTC

Contents

arl_html_vignette	2
cli	2
Engine	3
install_cli	8
register_knitr_engine	9

Index	10
--------------	-----------

arl_html_vignette	<i>HTML vignette format with Arl syntax highlighting</i>
-------------------	--

Description

A wrapper around `html_vignette` that passes `--syntax-definition` to Pandoc so that `{arl}` code blocks receive proper syntax highlighting.

Usage

```
arl_html_vignette(pandoc_args = NULL, check_title = TRUE, ...)
```

Arguments

<code>pandoc_args</code>	Additional Pandoc arguments (merged with the <code>syntax-definition</code> argument added automatically).
<code>check_title</code>	Whether to check that the vignette title matches the <code>VignetteIndexEntry</code> . Set to <code>FALSE</code> for pkgdown-only articles that intentionally omit a <code>VignetteIndexEntry</code> .
<code>...</code>	Arguments passed to <code>html_vignette</code> .

Value

An R Markdown output format object.

cli	<i>Run the Arl CLI</i>
-----	------------------------

Description

Entry point for the Arl command-line interface. Parses arguments and runs the requested action (REPL, file evaluation, or expression evaluation).

Usage

```
cli(args = commandArgs(trailingOnly = TRUE))
```

Arguments

args Command-line arguments to parse (defaults to `commandArgs(trailingOnly = TRUE)`).

Value

Invisibly returns `NULL`.

Engine

Core Arl engine

Description

Provides class-based organization for tokenization, parsing, macro expansion, evaluation, and environment management.

Methods**Public methods:**

- `Engine$new()`
- `Engine$read()`
- `Engine$write()`
- `Engine$eval()`
- `Engine$eval_text()`
- `Engine$eval_string()`
- `Engine$load_file_in_env()`
- `Engine$macroexpand()`
- `Engine$inspect_compilation()`
- `Engine$help()`
- `Engine$repl()`
- `Engine$enable_coverage()`
- `Engine$disable_coverage()`
- `Engine$get_coverage()`
- `Engine$reset_coverage()`
- `Engine$get_env()`
- `Engine$define()`
- `Engine$format_value()`
- `Engine$with_error_context()`
- `Engine$print_error()`

Method `new()`: Initialize engine components and base environment.

Usage:

```

Engine$new(
  coverage_tracker = NULL,
  load_prelude = TRUE,
  disable_tco = NULL,
  disable_constant_folding = NULL,
  disable_optimizations = NULL,
  disable_arithmetic_infix = NULL,
  disable_bytecode = NULL,
  r_packages = "search_path"
)

```

Arguments:

`coverage_tracker` Optional CoverageTracker instance to enable coverage tracking from the start. If provided, coverage will be tracked during stdlib loading. Intended for internal development use.

`load_prelude` Logical. If TRUE (the default), loads prelude modules during initialization. Set to FALSE to create a bare engine with only builtins — useful for testing or when you want to import specific modules.

`disable_tco` Optional logical. If TRUE, disables self-tail-call optimization in the compiler, preserving natural call stacks for debugging. Defaults to NULL, which inherits from global option `getOption("ar1.disable_tco", FALSE)`.

`disable_constant_folding` Optional logical. If TRUE, disables compile-time constant folding, forcing all expressions to be evaluated at runtime. Useful for testing that builtins match R semantics. Defaults to NULL, which inherits from global option `getOption("ar1.disable_constant_folding", FALSE)`.

`disable_optimizations` Optional logical. If TRUE, disables all non-essential compiler optimizations (constant folding, TCO, dead code elimination, strength reduction, identity elimination, truthiness optimization, begin simplification, and boolean flattening). Individual toggles like `disable_tco` and `disable_constant_folding` are applied after this and can override it. Defaults to NULL, which inherits from global option `getOption("ar1.disable_optimizations", FALSE)`.

`disable_arithmetic_infix` Logical; if TRUE, disable 2-arg arithmetic infix compilation.

`disable_bytecode` Logical; if TRUE, disable bytecode compilation of cached modules.

`r_packages` Controls which R packages are visible to Arl code. "search_path" (default) tracks R's `search()` dynamically; a character vector pins a fixed set; NULL exposes only `baseenv()`.

Method `read()`: Tokenize and parse source into expressions. The format returned by this method is not guaranteed to be stable across package versions.

Usage:

```
Engine$read(source, source_name = NULL)
```

Arguments:

`source` Character string containing Arl source.

`source_name` Optional source name for error reporting.

Returns: A list of parsed Arl expressions (R language objects).

Method `write()`: Convert an Arl expression to its string representation. Inverse of `read()`. The format returned by this method is not guaranteed to be stable across package versions.

Usage:

```
Engine$write(expr)
```

Arguments:

expr Arl expression (symbol/call/atomic value).

Returns: A character string of the Arl source representation.

Method eval(): Evaluate one or more expressions.

Usage:

```
Engine$eval(expr, ..., env = NULL)
```

Arguments:

expr Arl expression (symbol/call/atomic value).

... Additional Arl expressions to evaluate (variadic).

env Optional environment or Env used as the engine base.

Returns: The result of the last evaluated expression.

Method eval_text(): Read and evaluate Arl source text. Convenience wrapper around read() and eval().

Usage:

```
Engine$eval_text(text, env = NULL, source_name = "<eval>")
```

Arguments:

text Character string of Arl code to read/eval.

env Optional environment or Env used as the engine base.

source_name Optional source name for error reporting.

Returns: The result of the last evaluated expression.

Method eval_string(): Alias for eval_text().

Usage:

```
Engine$eval_string(text, env = NULL, source_name = "<eval>")
```

Arguments:

text Character string of Arl code to read/eval.

env Optional environment or Env used as the engine base.

source_name Optional source name for error reporting.

Returns: The result of the last evaluated expression.

Method load_file_in_env(): Load and evaluate an Arl source file in the given environment. Definitions and imports in the file are visible in env. To evaluate in an isolated child scope, create one explicitly: load_file_in_env(path, new.env(parent = env)).

Usage:

```
Engine$load_file_in_env(path, env = NULL, cache = TRUE)
```

Arguments:

path File path to load.

env Optional environment or Env used as the engine base.

cache Logical; if TRUE (the default), use the module cache.

Returns: The result of the last evaluated expression (invisibly).

Method `macroexpand()`: Expand macros in an expression. With `depth = NULL` (the default), fully and recursively expand all macros. With `depth = N`, expand only the top-level macro up to `N` times without walking into subexpressions.

Usage:

```
Engine$macroexpand(expr, env = NULL, depth = NULL, preserve_src = FALSE)
```

Arguments:

`expr` Arl expression (symbol/call/atomic value).

`env` Optional environment or Env used as the engine base.

`depth` Number of expansion steps (NULL for full expansion).

`preserve_src` Logical; keep source metadata when macroexpanding.

Method `inspect_compilation()`: Inspect expansion and compilation for debugging. Parse text, expand macros in `env`, then compile to R. Returns parsed AST, expanded form, compiled R expression, and deparsed R code so you can see exactly what an Arl program becomes.

Usage:

```
Engine$inspect_compilation(text, env = NULL, source_name = "<inspect>")
```

Arguments:

`text` Character; Arl source (single expression or multiple).

`env` Environment or NULL (use engine `env`). Must have `macros/stdlib` if needed.

`source_name` Name for parse errors.

Returns: List with parsed (first `expr`), expanded, compiled (R `expr` or NULL), `compiled_deparsed` (character, or NULL).

Method `help()`: Show help for a topic.

Usage:

```
Engine$help(topic, env = NULL, package = NULL)
```

Arguments:

`topic` Help topic as a single string.

`env` Optional environment/Env to resolve Arl bindings against.

`package` Optional package name (string or symbol) to force R help lookup in a specific package.

Returns: Help text (invisibly), or NULL if topic not found.

Method `repl()`: Start the Arl REPL using this engine.

Usage:

```
Engine$repl()
```

Returns: NULL (invisibly); called for side effects.

Method `enable_coverage()`: Enable coverage tracking.

Creates a coverage tracker and installs it in the eval context. Should be called before running code you want to track.

Usage:

Engine\$enable_coverage()

Returns: The engine (invisibly), for method chaining.

Method disable_coverage(): Disable coverage tracking.

Usage:

Engine\$disable_coverage()

Returns: The engine (invisibly), for method chaining.

Method get_coverage(): Get coverage data as a data frame.

Usage:

Engine\$get_coverage()

Returns: A data frame with columns file, total_lines, covered_lines, and coverage_pct (one row per tracked file), with a "total" attribute containing aggregate stats. Returns NULL if coverage tracking is not enabled.

Method reset_coverage(): Reset coverage data.

Usage:

Engine\$reset_coverage()

Returns: The engine (invisibly), for method chaining.

Method get_env(): Get the top-level R environment backing this engine.

Usage:

Engine\$get_env()

Returns: An R environment.

Method define(): Define a binding in the engine's top-level environment. This is the supported way to inject R objects for use in Arl code.

Usage:

Engine\$define(name, value)

Arguments:

name Character string; the binding name.

value The value to bind.

Returns: Invisible self (for chaining).

Method format_value(): Format a value for display using the engine's formatter.

Usage:

Engine\$format_value(value)

Arguments:

value Value to format.

Returns: Character string.

Method with_error_context(): Run a function with source-tracking error context.

Usage:

```
Engine$with_error_context(fn)
```

Arguments:

fn A zero-argument function to call.

Returns: The return value of fn.

Method print_error(): Format and print an Arl error with source context.

Usage:

```
Engine$print_error(e, file = stderr())
```

Arguments:

e A condition object.

file Connection to print to (default stderr()).

Returns: NULL (invisibly); called for side effects.

Examples

```
engine <- Engine$new()
engine$eval_text("+ 1 2 3")
engine$eval_string("+ 4 5")
```

install_cli

Install the Arl CLI wrapper

Description

Prints platform-appropriate instructions for making the packaged CLI wrapper available from the shell. No files are written or copied – the user follows the printed instructions themselves.

Usage

```
install_cli(quiet = FALSE)
```

Arguments

quiet If TRUE, suppress printed instructions and return the script path invisibly.

Value

The path to the CLI wrapper script, invisibly.

register_knitr_engine *Register the Arl knitr engine*

Description

Call this function in a vignette's setup chunk to enable {ar1} code chunks. The engine evaluates Arl code using a shared Engine instance that persists across chunks within a single document.

Usage

```
register_knitr_engine()
```

Value

NULL (invisibly); called for its side effect of registering the engine.

Examples

```
# In a vignette setup chunk:  
ar1::register_knitr_engine()
```

Index

`ar1_html_vignette`, [2](#)

`cli`, [2](#)

`Engine`, [3](#)

`html_vignette`, [2](#)

`install_cli`, [8](#)

`register_knitr_engine`, [9](#)