

Package: tccquickr (via r-universe)

October 11, 2026

Title Typed Experimental R Transformation Core

Version 0.0.0.9000

Description A hard-reset experimental compiler core for a declared subset of R. The package now starts from typed schemas, classed diagnostics, explicit compiler contracts, shape-aware intermediate representations, and generic backend planning. Initial source printers consume a backend-neutral lowered program rather than owning compiler semantics.

License GPL (>= 3)

Depends R (>= 4.4.0)

Imports codetools, S7, s7contract

Suggests Rtinycc, tinytest

Additional_repositories <https://sounkou-bioinfo.r-universe.dev>

Encoding UTF-8

Roxygen list(markdown = TRUE)

URL <https://github.com/sounkou-bioinfo/tccquickr>

BugReports <https://github.com/sounkou-bioinfo/tccquickr/issues>

Config/roxygen2/version 8.0.0

Repository <https://sounkou-bioinfo.r-universe.dev>

Date/Publication 2026-07-16 17:08:24 UTC

RemoteUrl <https://github.com/sounkou-bioinfo/tccquickr>

RemoteRef HEAD

RemoteSha d70ac0ed608bdb755fe4c9c821aa33934f640013

Contents

tccq_access	6
tccq_analyze	7
tccq_arg_reduction_spec	7

tccq_arity	8
tccq_arity_accepts	8
tccq_assignment	9
tccq_backend_allocation_binding	9
tccq_backend_artifact	10
tccq_backend_context	10
tccq_backend_error_channel	11
tccq_backend_extent_binding	11
tccq_backend_function_interface	12
tccq_backend_plan	13
tccq_backend_plan_set	14
tccq_backend_prepare	14
tccq_backend_products	15
tccq_backend_spec	15
tccq_backend_value_binding	16
tccq_binding	17
tccq_binding_reference	17
tccq_branch	18
tccq_bridge_plan	18
tccq_c_backend	19
tccq_call	20
tccq_call_index	20
tccq_call_name	21
tccq_call_semantics	21
tccq_cell	22
tccq_cell_reference	22
tccq_collect_call_index	23
tccq_collect_call_semantics	23
tccq_collect_calls	24
tccq_compile	24
tccq_completion	25
tccq_condition_diagnostic	25
tccq_conditional	26
tccq_contraction_spec	26
tccq_core_backends	27
tccq_default_op_registry	27
tccq_diagnostic	27
tccq_dim_affine	28
tccq_dim_constant	28
tccq_dim_equal	29
tccq_dim_symbol	29
tccq_dim_unknown	29
tccq_dimension_reference	30
tccq_domain	30
tccq_domain_policy	31
tccq_domain_policy_result_shape	31
tccq_effect	32
tccq_effect_union	32

tccq_elementwise_domain_policy	33
tccq_elementwise_expression	33
tccq_elementwise_result_type	34
tccq_elementwise_spec	34
tccq_evaluation_step	35
tccq_expression	35
tccq_expression_reference	36
tccq_expression_tree	37
tccq_for	37
tccq_fortran_backend	38
tccq_fusion_contract	38
tccq_fusion_group	39
tccq_if	40
tccq_index_expr	40
tccq_index_proof	41
tccq_indexed_value	41
tccq_iteration_plan	42
tccq_iteration_spec	43
tccq_literal	43
tccq_literal_finite	44
tccq_literal_inf	44
tccq_literal_na	44
tccq_literal_nan	45
tccq_local_binding	45
tccq_loop_axis	46
tccq_loop_guard	46
tccq_loop_nest	47
tccq_loop_transfer	47
tccq_lower_function	48
tccq_lowered_operation	48
tccq_op_body	49
tccq_op_context	50
tccq_op_impl	50
tccq_op_registry	51
tccq_op_registry_add	52
tccq_op_render	52
tccq_op_render_context	53
tccq_op_signature	53
tccq_op_signature_result_type	54
tccq_op_supports	54
tccq_opaque_op_impl	54
tccq_optimize	55
tccq_plan_backend	55
tccq_plan_backends	56
tccq_program	56
tccq_program_loop_nest	57
tccq_program_loop_nests	57
tccq_program_schedule	58

tccq_reduction_identity	58
tccq_reduction_plan	59
tccq_reduction_spec	59
tccq_reduction_state_component	60
tccq_region	60
tccq_registry_supports	61
tccq_repeat	62
tccq_resolve_call	62
tccq_resolved_op	63
tccq_result	63
tccq_rtinycc_backend	63
tccq_shape	64
tccq_storage_allocation	64
tccq_storage_lifetime	65
tccq_storage_plan	65
tccq_storage_slot	66
tccq_subscript_spec	66
tccq_switch	67
tccq_type	67
tccq_unimplemented_calls	68
tccq_value	68
tccq_value_block	69
tccq_verify_optimization	69
tccq_while	70
tccq_write_target	70
TccqAccess	71
TccqArgReductionSpec	71
TccqArity	72
TccqAssignment	73
TccqBackend	73
TccqBackendAllocationBinding	74
TccqBackendArtifact	74
TccqBackendContext	75
TccqBackendErrorChannel	76
TccqBackendExtentBinding	76
TccqBackendFunctionInterface	77
TccqBackendPlan	78
TccqBackendPlanSet	79
TccqBackendProducts	79
TccqBackendSpec	80
TccqBackendValueBinding	81
TccqBinding	82
TccqBindingReference	82
TccqBlock	83
TccqBranch	83
TccqBridgePlan	84
TccqCall	85
TccqCallIndex	86

TccqCallSemantics	86
TccqCell	87
TccqCellReference	88
TccqConditional	88
TccqContractionSpec	89
TccqControlCompletion	90
TccqDeadStoreOptimization	90
TccqDiagnostic	91
TccqDim	91
TccqDimensionReference	92
TccqDomain	92
TccqDomainPolicy	93
TccqEffect	93
TccqElementwiseSpec	94
TccqEvaluationStep	94
TccqExpression	95
TccqExpressionReference	96
TccqFoldReductionSpec	97
TccqFor	97
TccqFusionContract	98
TccqFusionGroup	99
TccqIf	100
TccqIndexedValue	101
TccqIndexExpr	102
TccqIndexProof	102
TccqIterationPlan	103
TccqIterationSpec	104
TccqLiteral	104
TccqLocalBinding	105
TccqLoop	105
TccqLoopAxis	106
TccqLoopGuard	106
TccqLoopNest	107
TccqLoopTransfer	108
TccqLoweredOperation	108
TccqLoweringPlan	109
TccqOpBody	110
TccqOpContext	110
TccqOpImpl	111
TccqOpImplementation	112
TccqOpRegistry	112
TccqOpRenderContext	113
TccqOpSignature	113
TccqProgram	114
TccqProgramBody	115
TccqProgramOptimization	115
TccqProgramSchedule	115
TccqReductionPlan	116

TccqReductionSpec	116
TccqReductionState	117
TccqReductionStateComponent	118
TccqRegion	118
TccqRepeat	119
TccqResolvedOp	120
TccqResult	121
TccqShape	121
TccqStatement	122
TccqStorageAllocation	122
TccqStorageLifetime	123
TccqStoragePlan	123
TccqStorageSlot	124
TccqSubscriptSpec	124
TccqSwitch	125
TccqType	126
TccqValue	126
TccqValueBlock	127
TccqValueGraph	127
TccqWhile	128
TccqWriteTarget	128

Index 130

tccq_access	<i>Construct a domain access mapping</i>
-------------	--

Description

Construct a domain access mapping

Usage

```
tccq_access(
  value_id,
  domain,
  kind = "identity",
  index_map = list(),
  consumer_shape = NULL
)
```

Arguments

value_id	Referenced value id.
domain	Access domain.
kind	Access kind.
index_map	Structured index-map payload.
consumer_shape	Logical consumer shape for recycling, or NULL for other access kinds.

tccq_analyze	Analyze a declared R function into the fresh program schema
--------------	---

Description

This is intentionally narrow. It parses `declare(type(...))` declarations, records a program schema, reports calls without implementations as classed diagnostics, and runs the first backend-neutral expression lowering pass when frontend facts are clean.

Usage

```
tccq_analyze(
  fn,
  strict = FALSE,
  registry = tccq_default_op_registry(),
  context = tccq_op_context()
)
```

Arguments

fn	Function to analyze.
strict	If TRUE, throw the first diagnostic as a classed condition.
registry	Operation registry used to decide whether calls are supported.
context	Operation support context.

tccq_arg_reduction_spec	Construct argument-selection reduction metadata
-------------------------	---

Description

Construct argument-selection reduction metadata

Usage

```
tccq_arg_reduction_spec(
  name,
  signature,
  direction = "max",
  missing = "ignore",
  ties = "first",
  attrs = list()
)
```

Arguments

name	Human-readable reduction name.
signature	Shared operation signature.
direction	Value-selection direction.
missing	Missing-value policy.
ties	Tie-selection policy.
attrs	Structured reduction metadata.

tccq_arity	<i>Construct an operation arity contract</i>
------------	--

Description

Construct an operation arity contract

Usage

tccq_arity(counts = integer(), minimum = NULL, maximum = NULL)

Arguments

counts	Exact accepted argument counts.
minimum	Inclusive lower argument-count bound, or NULL for an exact contract.
maximum	Inclusive upper argument-count bound, or NULL for no upper bound.

tccq_arity_accepts	<i>Test an operation arity contract</i>
--------------------	---

Description

Test an operation arity contract

Usage

tccq_arity_accepts(arity, count)

Arguments

arity	A TccqArity contract.
count	Observed argument count.

Value

One logical value.

tccq_assignment	<i>Construct a neutral assignment</i>
-----------------	---------------------------------------

Description

Construct a neutral assignment

Usage

tccq_assignment(id, target, value)

Arguments

- | | |
|--------|-----------------------------------|
| id | Stable statement id. |
| target | Typed destination. |
| value | Backend-neutral value expression. |

tccq_backend_allocation_binding	<i>Construct a backend allocation binding</i>
---------------------------------	---

Description

Construct a backend allocation binding

Usage

tccq_backend_allocation_binding(source_name, allocation, slots)

Arguments

- | | |
|-------------|--|
| source_name | Generated source identifier. |
| allocation | Physical allocation represented by the identifier. |
| slots | Materialized temporary slots sharing the allocation. |

tccq_backend_artifact *Construct a backend artifact*

Description

Construct a backend artifact

Usage

```
tccq_backend_artifact(  
    id,  
    kind,  
    source_language = "",  
    path = "",  
    attrs = list()  
)
```

Arguments

id	Stable artifact id.
kind	Artifact kind.
source_language	Source language used to produce the artifact, or empty.
path	Filesystem path for file-backed artifacts, or empty.
attrs	Structured artifact metadata.

tccq_backend_context *Construct a backend planning context*

Description

Construct a backend planning context

Usage

```
tccq_backend_context(  
    mode = "source",  
    target = "any",  
    allow_boundary = FALSE,  
    required_capabilities = character(),  
    attrs = list()  
)
```

Arguments

mode	Requested backend mode.
target	Requested target language or runtime, or any.
allow_boundary	Whether explicit R call-evaluation boundary plans are allowed.
required_capabilities	Capabilities that candidate backends must expose.
attrs	Structured backend-context attributes.

tccq_backend_error_channel

Construct a backend callable error channel

Description

Construct a backend callable error channel

Usage

```
tccq_backend_error_channel(source_name, diagnostics)
```

Arguments

source_name	Generated status-output parameter name.
diagnostics	Ordered runtime diagnostics addressable by status code.

tccq_backend_extent_binding

Construct a backend extent binding

Description

Construct a backend extent binding

Usage

```
tccq_backend_extent_binding(source_name, symbol)
```

Arguments

source_name	Generated extent parameter name.
symbol	Declared dimension symbol.

tccq_backend_function_interface

Construct a backend function interface

Description

Construct a backend function interface

Usage

```
tccq_backend_function_interface(
    symbol,
    source_language,
    kind,
    abi = "",
    parameters = list(),
    locals = list(),
    allocations = list(),
    result_value_id,
    result_type,
    result_placement = "return",
    result_name = "",
    domain = NULL,
    extents = list(),
    index_names = character(),
    result_dims = list(),
    result_count_name = "",
    error_channel = NULL
)
```

Arguments

symbol	Generated function symbol.
source_language	Source language consumed by the printer.
kind	Function shape: scalar, loop_nest, or structured.
abi	Callable ABI.
parameters	Ordered ABI parameter bindings.
locals	Ordered scalar-local bindings.
allocations	Ordered physical-allocation bindings.
result_value_id	Lowered result value id.
result_type	Declared semantic result type.
result_placement	Whether the result is returned or passed by output argument.

result_name	Generated result/output variable name, or empty string.
domain	Loop-nest iteration domain, or NULL.
extents	Ordered symbolic-extent bindings carried by the ABI.
index_names	Generated data-parallel and structured loop index names.
result_dims	Typed result dimensions.
result_count_name	Generated result element-count parameter name, or empty string for scalar results.
error_channel	Optional generated callable error channel.

tccq_backend_plan	<i>Construct a backend plan</i>
-------------------	---------------------------------

Description

Construct a backend plan

Usage

```
tccq_backend_plan(
    id,
    backend_id,
    family,
    mode,
    target,
    capabilities = character(),
    regions = list(),
    bridges = list(),
    diagnostics = list(),
    products = NULL,
    attrs = list()
)
```

Arguments

id	Stable plan id.
backend_id	Backend spec id.
family	Backend family.
mode	Backend mode.
target	Target language or runtime.
capabilities	Backend capabilities visible to this plan.
regions	Regions assigned to the backend plan.
bridges	Representation-transition plans.

diagnostics	Diagnostics attached to this plan.
products	Typed backend products.
attrs	Structured plan attributes.

tccq_backend_plan_set	<i>Construct a backend plan set</i>
-----------------------	-------------------------------------

Description

Construct a backend plan set

Usage

```
tccq_backend_plan_set(  
    program_name,  
    plans = list(),  
    diagnostics = list(),  
    attrs = list()  
)
```

Arguments

program_name	Program name.
plans	Backend plans.
diagnostics	Diagnostics collected while planning.
attrs	Structured plan-set attributes.

tccq_backend_prepare	<i>Prepare a backend plan</i>
----------------------	-------------------------------

Description

Prepare a backend plan

Usage

```
tccq_backend_prepare(backend, program, context)
```

Arguments

backend	Backend implementation.
program	Program to plan.
context	Backend planning context.

tccq_backend_products	<i>Construct backend products</i>
-----------------------	-----------------------------------

Description

Construct backend products

Usage

```
tccq_backend_products(  
    function_interface = NULL,  
    body = NULL,  
    loop_nest = NULL,  
    loop_nests = list(),  
    storage_plan = NULL,  
    artifacts = list(),  
    attrs = list()  
)
```

Arguments

function_interface	Source-level callable interface, if source exists.
body	Expression tree or statement block consumed by the backend, if source exists.
loop_nest	Result loop nest consumed by the backend, if source exists.
loop_nests	Ordered loop nests, intermediates first, result nest last.
storage_plan	Storage plan consumed by the backend, if available.
artifacts	Named backend artifacts.
attrs	Structured product metadata.

tccq_backend_spec	<i>Construct a backend implementation descriptor</i>
-------------------	--

Description

Construct a backend implementation descriptor

Usage

```
tccq_backend_spec(
  id,
  family,
  target,
  driver,
  modes,
  region_kinds,
  memory_spaces,
  capabilities,
  uses_rapi,
  attrs = list(),
  prepare = NULL
)
```

Arguments

id	Stable backend id.
family	Backend family.
target	Target language or runtime.
driver	Concrete driver name.
modes	Supported backend modes.
region_kinds	Supported region kinds.
memory_spaces	Supported memory spaces.
capabilities	Capabilities exposed by this backend.
uses_rapi	Whether this backend may touch the R C API.
attrs	Structured backend attributes.
prepare	Planning implementation.

tccq_backend_value_binding

Construct a backend value binding

Description

Construct a backend value binding

Usage

```
tccq_backend_value_binding(source_name, value_id, source_type, role)
```

Arguments

source_name	Generated source identifier.
value_id	Neutral value id.
source_type	Type represented by the generated identifier.
role	Whether the identifier is an ABI parameter or scalar local.

tccq_binding	<i>Construct a program binding</i>
--------------	------------------------------------

Description

Construct a program binding

Usage

```
tccq_binding(name, type, mutable = FALSE)
```

Arguments

name	Binding name.
type	Binding type.
mutable	Whether the binding may be mutated.

tccq_binding_reference	<i>Construct a local binding reference</i>
------------------------	--

Description

Construct a local binding reference

Usage

```
tccq_binding_reference(id, binding)
```

Arguments

id	Stable reference value id.
binding	Local binding being read.

tccq_branch	<i>Construct a lazy conditional IR value</i>
-------------	--

Description

Construct a lazy conditional IR value

Usage

```
tccq_branch(  
  id,  
  condition,  
  consequent,  
  alternative,  
  type,  
  semantics,  
  effect = tccq_effect(may_error = TRUE),  
  attrs = list()  
)
```

Arguments

id	Stable value id.
condition	Scalar logical condition value id.
consequent	Value id selected when the condition is true.
alternative	Value id selected when the condition is false.
type	Joined branch result type.
semantics	Evaluator facts for the originating if call.
effect	Conservative effect summary across the condition and both arms.
attrs	Structured branch metadata.

tccq_bridge_plan	<i>Construct a backend bridge plan</i>
------------------	--

Description

Construct a backend bridge plan

Usage

```
tccq_bridge_plan(  
    id,  
    kind,  
    from_space,  
    to_space,  
    from_type = NULL,  
    to_type = NULL,  
    effect = tccq_effect(reads = TRUE),  
    attrs = list()  
)
```

Arguments

id	Stable bridge id.
kind	Bridge kind.
from_space	Source memory space.
to_space	Destination memory space.
from_type	Optional source type.
to_type	Optional destination type.
effect	Bridge effect summary.
attrs	Structured bridge attributes.

tccq_c_backend	<i>Generic C backend descriptor</i>
----------------	-------------------------------------

Description

Generic C backend descriptor

Usage

```
tccq_c_backend()
```

tccq_call	<i>Construct an observed call</i>
-----------	-----------------------------------

Description

Construct an observed call

Usage

```
tccq_call(  
  name,  
  expr = NULL,  
  origin = "ast",  
  id = "",  
  kind = NULL,  
  arity = NULL,  
  argument_names = NULL,  
  attrs = list()  
)
```

Arguments

- | | |
|----------------|---|
| name | Call name. |
| expr | Original R call expression. |
| origin | Source of the call observation. |
| id | Stable call id. |
| kind | Structural call kind. |
| arity | Number of supplied arguments, or NA_integer_. |
| argument_names | Supplied argument tags. |
| attrs | Structured call attributes. |

tccq_call_index	<i>Construct a typed call index</i>
-----------------	-------------------------------------

Description

Construct a typed call index

Usage

```
tccq_call_index(calls, semantics = NULL, env = baseenv(), attrs = list())
```

Arguments

calls	List of observed calls.
semantics	Optional list of call evaluator facts. If omitted, inferred.
env	Environment used when inferring missing semantics.
attrs	Structured index attributes.

tccq_call_name	<i>Return the name of an R call</i>
----------------	-------------------------------------

Description

Return the name of an R call

Usage

```
tccq_call_name(call)
```

Arguments

call	R call.
------	---------

tccq_call_semantics	<i>Construct call evaluator facts</i>
---------------------	---------------------------------------

Description

Construct call evaluator facts

Usage

```
tccq_call_semantics(
  call,
  env = baseenv(),
  evaluator_kind = NULL,
  forcing_policy = NULL,
  dispatch_kind = NULL,
  lexical_scope = NULL,
  replacement = NULL,
  control = NULL,
  attrs = list()
)
```

Arguments

call	Observed call.
env	Environment used to resolve ordinary function names.
evaluator_kind	Optional evaluator kind override.
forcing_policy	Optional forcing-policy override.
dispatch_kind	Optional dispatch-kind override.
lexical_scope	Optional lexical-scope override.
replacement	Optional replacement flag override.
control	Optional control flag override.
attrs	Structured semantic attributes.

tccq_cell	<i>Construct a mutable lexical cell</i>
-----------	---

Description

Construct a mutable lexical cell

Usage

```
tccq_cell(name, value_id, type, storage_type = tccq_type(type@base))
```

Arguments

name	Binding name.
value_id	Stable storage identity.
type	Binding type.
storage_type	Scalar storage type.

tccq_cell_reference	<i>Construct a lowered mutable-cell read</i>
---------------------	--

Description

Construct a lowered mutable-cell read

Usage

```
tccq_cell_reference(id, cell)
```

Arguments

id	Stable value id for this read.
cell	Mutable cell being read.

`tccq_collect_call_index`*Collect a typed call index from an R expression*

Description

Collect a typed call index from an R expression

Usage

```
tccq_collect_call_index(  
  expr,  
  global_calls = character(),  
  env = baseenv(),  
  attrs = list()  
)
```

Arguments

<code>expr</code>	R expression to inspect.
<code>global_calls</code>	Additional function names found by codetools.
<code>env</code>	Environment used to resolve ordinary function names.
<code>attrs</code>	Structured index attributes.

`tccq_collect_call_semantics`*Collect evaluator facts for calls*

Description

Collect evaluator facts for calls

Usage

```
tccq_collect_call_semantics(calls, env = baseenv())
```

Arguments

<code>calls</code>	List of <code>TccqCall</code> objects.
<code>env</code>	Environment used to resolve ordinary function names.

tccq_collect_calls	<i>Collect calls from an R expression</i>
--------------------	---

Description

Collect calls from an R expression

Usage

```
tccq_collect_calls(expr, global_calls = character())
```

Arguments

expr	R expression to inspect.
global_calls	Additional function names found by codetools.

tccq_compile	<i>Compile a declared R function</i>
--------------	--------------------------------------

Description

Compilation applies one explicit typed optimization, then asks the core backend suite to account for the same verified program. Generic C, Rtinycc/TinyCC, and quickr-style Fortran report constraints through one contract instead of letting one concrete backend shape the IR. The result succeeds when at least one backend produces a working plan; backends that cannot lower the program report feasibility diagnostics in their plans without vetoing the suite.

Usage

```
tccq_compile(
  fn,
  backends = tccq_core_backends(),
  context = tccq_backend_context(),
  optimization = TccqDeadStoreOptimization(),
  strict = TRUE
)
```

Arguments

fn	Function to compile.
backends	Backend implementation descriptors.
context	Backend planning context.
optimization	Program optimization applied before backend planning.
strict	If TRUE, throw the first diagnostic as a classed condition.

tccq_completion	<i>Compute structured control completion</i>
-----------------	--

Description

Completion summarizes only paths reachable in evaluation order. Nested loops consume their own break and next transfers, while a block retains transfers that target its nearest enclosing loop.

Usage

tccq_completion(control)

Arguments

control Typed statement or statement block.

Value

A TccqControlCompletion value.

tccq_condition_diagnostic	<i>Extract a diagnostic from a compiler condition</i>
---------------------------	---

Description

Extract a diagnostic from a compiler condition

Usage

tccq_condition_diagnostic(condition)

Arguments

condition A compiler condition.

tccq_conditional	<i>Construct a value-producing neutral conditional</i>
------------------	--

Description

Construct a value-producing neutral conditional

Usage

```
tccq_conditional(id, condition, consequent, alternative, branch)
```

Arguments

id	Stable statement id.
condition	Scalar logical condition expression.
consequent	Block evaluated when the condition is true.
alternative	Block evaluated when the condition is false.
branch	Source branch payload retaining R special-form semantics.

tccq_contraction_spec	<i>Construct contraction implementation metadata</i>
-----------------------	--

Description

Construct contraction implementation metadata

Usage

```
tccq_contraction_spec(
  name,
  signature,
  reducer,
  combine_op = "*",
  attrs = list()
)
```

Arguments

name	Human-readable contraction name.
signature	Shared operation signature.
reducer	Reduction metadata used for the contracted axes.
combine_op	Elementwise operation name combining aligned elements.
attrs	Structured contraction metadata.

tccq_core_backends	<i>Core backend suite</i>
--------------------	---------------------------

Description

Every descriptor in the core suite prints from the shared loop nest and function interface. A backend family enters this suite together with its first real lowering; capability strings describe implemented behavior only.

Usage

```
tccq_core_backends(include_rtinycc = TRUE)
```

Arguments

include_rtinycc

Whether to include the Rtinycc TinyCC descriptor.

tccq_default_op_registry

Default operation registry for the reset frontend

Description

The default registry names the R language call forms that the frontend recognizes structurally. It is not a source-syntax whitelist and it is not a backend lowering promise. The registry is immutable, so it is built once per session and shared.

Usage

```
tccq_default_op_registry()
```

tccq_diagnostic

Create a compiler diagnostic

Description

Create a compiler diagnostic

Usage

```
tccq_diagnostic(code, message, phase = "schema", path = "", data = list())
```

Arguments

code	Stable diagnostic code.
message	Human-facing message.
phase	Compiler phase that produced the diagnostic.
path	Optional path into the program or schema.
data	Structured diagnostic payload.

tccq_dim_affine	<i>Construct an affine dimension</i>
-----------------	--------------------------------------

Description

An affine dimension is a symbolic extent plus a signed integer offset, such as the $n - 2$ extent of an interior stencil domain. It keeps shape arithmetic a typed dimension fact instead of a backend string.

Usage

tccq_dim_affine(name, offset)

Arguments

name	Symbolic dimension name.
offset	Signed integer offset added to the symbolic extent.

tccq_dim_constant	<i>Construct a constant dimension</i>
-------------------	---------------------------------------

Description

Construct a constant dimension

Usage

tccq_dim_constant(value)

Arguments

value	Non-negative integer dimension.
-------	---------------------------------

tccq_dim_equal	<i>Compare two dimensions for semantic equality</i>
----------------	---

Description

Constant dimensions compare by value, symbolic and affine dimensions by symbol and offset. Unknown dimensions never compare equal.

Usage

```
tccq_dim_equal(left, right)
```

Arguments

left, right TccqDim values.

tccq_dim_symbol	<i>Construct a symbolic dimension</i>
-----------------	---------------------------------------

Description

Construct a symbolic dimension

Usage

```
tccq_dim_symbol(name)
```

Arguments

name Dimension symbol.

tccq_dim_unknown	<i>Construct an unknown dimension</i>
------------------	---------------------------------------

Description

Construct an unknown dimension

Usage

```
tccq_dim_unknown()
```

tccq_dimension_reference	<i>Construct a typed declared-dimension reference</i>
--------------------------	---

Description

Construct a typed declared-dimension reference

Usage

tccq_dimension_reference(source_value_id, dimension)

Arguments

- | | |
|-----------------|--|
| source_value_id | Logical value id supplying the referenced storage. |
| dimension | Declared symbolic dimension supplied by the reference. |

tccq_domain	<i>Construct an iteration domain</i>
-------------	--------------------------------------

Description

Construct an iteration domain

Usage

tccq_domain(id, shape, axes = NULL, attrs = list())

Arguments

- | | |
|-------|--|
| id | Stable domain id. |
| shape | Domain shape. |
| axes | Optional axis names. Defaults to i1, i2, ... |
| attrs | Structured domain attributes. |

tccq_domain_policy	<i>Construct operation domain policy metadata</i>
--------------------	---

Description

Construct operation domain policy metadata

Usage

```
tccq_domain_policy(name, result_shape, attrs = list())
```

Arguments

- | | |
|--------------|--|
| name | Human-readable domain policy name. |
| result_shape | Function from input TccqType list to result TccqShape. |
| attrs | Structured domain-policy metadata. |

tccq_domain_policy_result_shape	<i>Return an operation domain-policy result shape</i>
---------------------------------	---

Description

Return an operation domain-policy result shape

Usage

```
tccq_domain_policy_result_shape(policy, input_types)
```

Arguments

- | | |
|-------------|--------------------------------|
| policy | Domain policy. |
| input_types | List of input TccqType values. |

tccq_effect	<i>Construct an effect summary</i>
-------------	------------------------------------

Description

Construct an effect summary

Usage

```
tccq_effect(
  reads = FALSE,
  writes = FALSE,
  allocates = FALSE,
  boundary = FALSE,
  may_error = FALSE,
  may_warn = FALSE
)
```

Arguments

reads	Whether the operation reads program state.
writes	Whether the operation writes program state.
allocates	Whether the operation allocates storage.
boundary	Whether the operation crosses an unsupported boundary.
may_error	Whether the operation may signal at runtime.
may_warn	Whether the operation may warn at runtime.

tccq_effect_union	<i>Combine effect summaries</i>
-------------------	---------------------------------

Description

Effect summaries are conservative may-properties. Combining two summaries retains every effect reported by either input.

Usage

```
tccq_effect_union(effect, other)
```

Arguments

effect	Effect summary.
other	Effect summary to combine with effect.

tccq_elementwise_domain_policy

Construct the standard elementwise domain policy

Description

The standard elementwise policy returns the shared non-scalar input shape and accepts scalar broadcasting. Following R's recycling rule, a shorter non-scalar operand is also accepted when its dimension multiset divides the longest operand's dimension multiset (so its total length provably divides the iteration size); the result takes the longest shape, and accesses to the shorter operand recycle over the iteration order. Shapes whose divisibility cannot be proven from declared dimensions are rejected.

Usage

```
tccq_elementwise_domain_policy()
```

tccq_elementwise_expression

Build a registered elementwise expression

Description

This transformation resolves one operation through the supplied registry, applies its signature to typed input expressions, and returns the ordinary neutral expression consumed by every source backend.

Usage

```
tccq_elementwise_expression(registry, op, inputs, id)
```

Arguments

registry	Operation registry used for resolution.
op	Operation name.
inputs	Ordered scalar or array input expressions.
id	Stable expression id.

Value

A `TccqResult` containing a `TccqExpression`.

<code>tccq_elementwise_result_type</code>
<i>Return an elementwise operation result type</i>

Description

Return an elementwise operation result type

Usage

`tccq_elementwise_result_type(spec, input_types)`

Arguments

- | | |
|--------------------------|--------------------------------|
| <code>spec</code> | Elementwise metadata. |
| <code>input_types</code> | List of input TccqType values. |

<code>tccq_elementwise_spec</code>	<i>Construct elementwise implementation metadata</i>
------------------------------------	--

Description

Construct elementwise implementation metadata

Usage

```
tccq_elementwise_spec(  
    name,  
    arity,  
    result_type,  
    attrs = list(),  
    domain_policy = NULL  
)
```

Arguments

- | | |
|----------------------------|---|
| <code>name</code> | Human-readable elementwise operation name. |
| <code>arity</code> | Accepted argument counts. |
| <code>result_type</code> | Function from input TccqType list to result TccqType. |
| <code>attrs</code> | Structured elementwise metadata. |
| <code>domain_policy</code> | Optional result-shape policy. |

tccq_evaluation_step *Construct a scheduled R evaluation*

Description

Construct a scheduled R evaluation

Usage

```
tccq_evaluation_step(
  value_id,
  statement_index,
  effect,
  binding = NULL,
  uses = list()
)
```

Arguments

value_id	Lowered value evaluated by this step.
statement_index	One-based executable statement position.
effect	Effect of evaluating the complete step.
binding	Optional local binding defined by the evaluation.
uses	Local bindings read by the evaluation.

tccq_expression *Construct a backend-neutral expression*

Description

Construct a backend-neutral expression

Usage

```
tccq_expression(
  id,
  kind,
  type,
  value_id = id,
  op = kind,
  inputs = list(),
  effect = tccq_effect(),
  literal = NULL,
```

```

    operation = NULL,
    branch = NULL,
    reference = NULL,
    index_proofs = list()
)

```

Arguments

id	Stable expression id.
kind	Expression kind.
type	Result type.
value_id	Lowered value id represented by this expression.
op	Operation name.
inputs	Child expressions.
effect	Effect of evaluating the expression.
literal	Literal payload for literal expressions.
operation	Typed lowered-operation payload for operation expressions.
branch	Typed conditional payload for branch expressions.
reference	Typed source payload for reference expressions.
index_proofs	Per-axis bounds proofs for indexed expressions.

tccq_expression_reference

Construct a typed expression reference

Description

Construct a typed expression reference

Usage

```

tccq_expression_reference(
  source_value_id,
  symbol = "",
  binding = NULL,
  slice_offsets = integer(),
  access = NULL
)

```

Arguments

source_value_id	Logical value id supplying the referenced storage.
symbol	Optional source symbol, or an empty string when not applicable.
binding	Optional lexical binding represented by the reference.
slice_offsets	Zero-based slice offsets, or an empty integer vector.
access	Optional typed domain access attached by loop-nest planning.

tccq_expression_tree	<i>Build a backend-neutral expression tree from a lowered value graph</i>
----------------------	---

Description

Build a backend-neutral expression tree from a lowered value graph

Usage

```
tccq_expression_tree(graph, value_id = NULL)
```

Arguments

graph	Lowered value graph.
value_id	Result value id to root the expression tree at.

tccq_for	<i>Construct a neutral for statement</i>
----------	--

Description

Construct a neutral for statement

Usage

```
tccq_for(id, iterator, iteration, body, semantics)
```

Arguments

id	Stable statement id.
iterator	Mutable scalar destination receiving each element.
iteration	Backend-neutral iteration plan.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.

tccq_fortran_backend *Quickr-style Fortran backend descriptor*

Description

This descriptor captures the pressure from quickr's Fortran path: subroutine lowering, C/R bridges, R CMD SHLIB, BLAS/LAPACK linkage, and optional OpenMP.

Usage

```
tccq_fortran_backend()
```

tccq_fusion_contract *Construct a fusion operation contract*

Description

Construct a fusion operation contract

Usage

```
tccq_fusion_contract(
  fusion_kind,
  result_value,
  operations = list(),
  storage_strategy = NULL,
  attrs = list()
)
```

Arguments

fusion_kind	Fusion kind owned by the contract.
result_value	Typed value produced by the fusion group.
operations	Named lowered operations by value id. Control-only groups may use an empty list.
storage_strategy	Optional storage strategy. Defaults from fusion_kind.
attrs	Structured metadata.

tccq_fusion_group	<i>Construct a fusion group</i>
-------------------	---------------------------------

Description

Construct a fusion group

Usage

```
tccq_fusion_group(
    id,
    kind,
    domain,
    values = list(),
    outputs = character(),
    accesses = list(),
    region_kind = "kernel",
    target = "any",
    effect = tccq_effect(),
    contract = NULL,
    attrs = list()
)
```

Arguments

id	Stable fusion-group id.
kind	Fusion kind.
domain	Shared iteration domain.
values	List of IR values in the group.
outputs	Output value ids.
accesses	List of access mappings.
region_kind	Candidate execution region kind, or any.
target	Candidate implementation target, or any.
effect	Effect summary for the fused group.
contract	Typed fusion operation/storage contract.
attrs	Structured fusion attributes.

tccq_if	<i>Construct a neutral procedural if</i>
---------	--

Description

Construct a neutral procedural if

Usage

```
tccq_if(id, condition, consequent, alternative, semantics)
```

Arguments

id	Stable statement id.
condition	Scalar logical condition expression.
consequent	Block evaluated when the condition is true.
alternative	Block evaluated when the condition is false.
semantics	Evaluator facts for the originating if special form.

tccq_index_expr	<i>Construct an affine index expression</i>
-----------------	---

Description

Construct an affine index expression

Usage

```
tccq_index_expr(axis = "", offset = 0L)
```

Arguments

axis	Iteration-axis name, or empty for a constant index.
offset	Signed integer offset (zero-based).

tccq_index_proof	<i>Construct a proven affine scalar index</i>
------------------	---

Description

Construct a proven affine scalar index

Usage

```
tccq_index_proof(  
    iterator,  
    selector,  
    iteration,  
    source_extent,  
    index,  
    operation = NULL,  
    semantics = NULL  
)
```

Arguments

- | | |
|---------------|--|
| iterator | Iterator target populated by the enclosing virtual loop. |
| selector | Direct reference to the iterator cell. |
| iteration | Virtual iteration establishing the selector range. |
| source_extent | Extent of the indexed source axis. |
| index | Zero-based affine storage index. |
| operation | Lowered selector operation for a shifted index, or NULL. |
| semantics | Evaluator facts for the shifted selector call, or NULL. |

tccq_indexed_value	<i>Construct a proven scalar indexed read</i>
--------------------	---

Description

Construct a proven scalar indexed read

Usage

```
tccq_indexed_value(
    id,
    source_id,
    source_type,
    index_proofs,
    access,
    operation,
    semantics
)
```

Arguments

id	Stable value id.
source_id	Stable value id of the source storage.
source_type	Type of the source storage.
index_proofs	Per-axis affine selector and bounds proofs.
access	Zero-based source access corresponding to the proofs.
operation	Lowered subscript implementation payload.
semantics	Evaluator facts for the originating [call.

tccq_iteration_plan	<i>Construct a backend-neutral iteration plan</i>
---------------------	---

Description

Construct a backend-neutral iteration plan

Usage

```
tccq_iteration_plan(
    source,
    domain,
    element,
    element_type,
    resolved_op = NULL,
    semantics = NULL
)
```

Arguments

source	Expression evaluated once to establish iteration.
domain	One-dimensional iteration domain.
element	Current element expression or affine induction value.

element_type	Scalar type assigned to the iteration cell.
resolved_op	Resolved operation for virtual iteration, or NULL for stored-value iteration.
semantics	Evaluator facts for the virtual iteration call, or NULL.

tccq_iteration_spec	<i>Construct iteration implementation metadata</i>
---------------------	--

Description

Construct iteration implementation metadata

Usage

tccq_iteration_spec(name, signature, extent_arg = 1L, start = 1L)

Arguments

name	Human-readable iteration operation name.
signature	Shared operation signature.
extent_arg	One-based argument position supplying the domain extent.
start	First integer value produced by the iteration.

tccq_literal	<i>Construct a scalar literal</i>
--------------	-----------------------------------

Description

Construct a scalar literal

Usage

tccq_literal(kind, type, value = NULL)

Arguments

kind	Literal kind: finite, na, nan, pos_inf, or neg_inf.
type	Scalar type.
value	Literal payload for finite values.

tccq_literal_finite	<i>Construct a finite scalar literal</i>
---------------------	--

Description

Construct a finite scalar literal

Usage

```
tccq_literal_finite(value, type = NULL)
```

Arguments

value	Finite scalar value.
type	Optional scalar type. If omitted, it is inferred from value.

tccq_literal_inf	<i>Construct an infinite double literal</i>
------------------	---

Description

Construct an infinite double literal

Usage

```
tccq_literal_inf(sign = 1L)
```

Arguments

sign	Positive or negative sign.
------	----------------------------

tccq_literal_na	<i>Construct a typed NA literal</i>
-----------------	-------------------------------------

Description

Construct a typed NA literal

Usage

```
tccq_literal_na(base)
```

Arguments

base	Base type for the missing value.
------	----------------------------------

tccq_literal_nan	<i>Construct a NaN literal</i>
------------------	--------------------------------

Description

Construct a NaN literal

Usage

tccq_literal_nan()

tccq_local_binding	<i>Construct a local value binding</i>
--------------------	--

Description

Construct a local value binding

Usage

tccq_local_binding(name, value_id, type, statement_index, mutable = FALSE)

Arguments

- | | |
|-----------------|---|
| name | Binding name. |
| value_id | Lowered value bound to the local name. |
| type | Binding type. |
| statement_index | One-based executable statement position defining the binding. |
| mutable | Whether the binding may be mutated. |

tccq_loop_axis	<i>Construct a loop-nest iteration axis</i>
----------------	---

Description

Construct a loop-nest iteration axis

Usage

```
tccq_loop_axis(name, extent, role = "map")
```

Arguments

name	Iteration-axis name.
extent	Typed axis extent.
role	Axis role, map or reduce.

tccq_loop_guard	<i>Construct a selected-arm loop guard</i>
-----------------	--

Description

Construct a selected-arm loop guard

Usage

```
tccq_loop_guard(condition, branch, selected)
```

Arguments

condition	Scalar logical condition expression.
branch	Source branch payload retaining R special-form semantics.
selected	Whether the consequent (TRUE) or alternative (FALSE) arm selects the guarded loop nest.

tccq_loop_nest	<i>Construct a loop nest</i>
----------------	------------------------------

Description

Construct a loop nest

Usage

```
tccq_loop_nest(  
    id,  
    axes,  
    body,  
    storage,  
    output = NULL,  
    reduction = NULL,  
    domain = NULL,  
    guards = list()  
)
```

Arguments

id	Stable loop-nest id.
axes	Ordered TccqLoopAxis values, outermost first.
body	Body expression or typed statement block with access-carrying references.
storage	Typed storage slot receiving the nest result.
output	Output access over map axes, or NULL for scalar results.
reduction	Closed neutral reduction plan for reduce axes, or NULL.
domain	Optional iteration domain. Defaults to one built from axes.
guards	Ordered TccqLoopGuard control path selecting this nest.

tccq_loop_transfer	<i>Construct a neutral loop transfer</i>
--------------------	--

Description

Construct a neutral loop transfer

Usage

```
tccq_loop_transfer(id, action, semantics)
```

Arguments

id	Stable statement id.
action	Loop action, either break or next.
semantics	Evaluator facts for the originating R control special form.

tccq_lower_function	<i>Lower a declared function into a backend-neutral plan</i>
---------------------	--

Description

The lowerer produces TccqValue, TccqRegion, TccqFusionGroup, and TccqStoragePlan objects. It is a middle-end pass result, not a C, Fortran, or TinyCC lowering.

Usage

```
tccq_lower_function(
  fn,
  bindings,
  registry = tccq_default_op_registry(),
  context = tccq_op_context(),
  call_index = NULL
)
```

Arguments

fn	Function to lower.
bindings	Declared formal bindings.
registry	Operation registry used for operation availability.
context	Operation support context.
call_index	Typed call facts for fn, or NULL to collect them.

tccq_lowered_operation	<i>Construct a lowered operation payload</i>
------------------------	--

Description

Construct a lowered operation payload

Usage

```
tccq_lowered_operation(
  family,
  resolved_op,
  signature = NULL,
  elementwise = NULL,
  reduction = NULL,
  contraction = NULL,
  subscript = NULL,
  attrs = list()
)
```

Arguments

family	Lowered operation family.
resolved_op	Selected operation implementation.
signature	Optional operation signature. Defaults to the family spec.
elementwise	Optional elementwise metadata.
reduction	Optional reduction metadata.
contraction	Optional contraction metadata.
subscript	Optional subscript metadata.
attrs	Structured metadata.

tccq_op_body

Construct a backend-neutral operation body

Description

fn supplies only ordered parameter names and one implementation expression; parameter types come from the operation signature at each expansion site. Exact call tags bind first and remaining arguments bind positionally. Partial tags, defaults, and . . . are excluded because their complete R argument and promise semantics are not represented by this expression-body contract.

Usage

```
tccq_op_body(fn)
```

Arguments

fn	Function with simple formals and one implementation expression.
----	---

tccq_op_context	<i>Construct an operation support context</i>
-----------------	---

Description

Construct an operation support context

Usage

```
tccq_op_context(
  phase = "frontend",
  target = "any",
  region_kind = "any",
  memory_space = "any",
  allow_rapi = TRUE,
  allow_boundary = FALSE
)
```

Arguments

phase	Compiler phase issuing the query.
target	Requested implementation target, or any.
region_kind	Requested execution region kind, or any.
memory_space	Requested memory space, or any.
allow_rapi	Whether implementations touching the R C API are allowed.
allow_boundary	Whether explicit boundary implementations are allowed.

tccq_op_impl	<i>Construct an operation implementation</i>
--------------	--

Description

Construct an operation implementation

Usage

```
tccq_op_impl(
  op,
  target,
  region_kind = "any",
  memory_space = "any",
  uses_rapi = FALSE,
  boundary = FALSE,
  pure = TRUE,
```

```

    effect = NULL,
    supports = function(call, context) TRUE,
    render = NULL,
    body = NULL,
    elementwise = NULL,
    reduction = NULL,
    contraction = NULL,
    iteration = NULL,
    subscript = NULL
  )

```

Arguments

op	Operation or function name.
target	Implementation target.
region_kind	Region kind the implementation can run in, or any.
memory_space	Memory space the implementation expects, or any.
uses_rapi	Whether the implementation touches the R C API.
boundary	Whether the implementation crosses a boundary.
pure	Whether the implementation is semantically pure.
effect	Effect summary for calls handled by this implementation.
supports	Predicate receiving a TccqCall and TccqOpContext.
render	Optional source renderer receiving operand strings and a TccqOpRenderContext.
body	Optional backend-neutral operation body.
elementwise	Optional elementwise metadata.
reduction	Optional reduction metadata.
contraction	Optional contraction metadata.
iteration	Optional iteration metadata.
subscript	Optional subscript metadata.

tccq_op_registry	<i>Construct an operation registry</i>
------------------	--

Description

Construct an operation registry

Usage

```
tccq_op_registry(implementations = list())
```

Arguments

implementations	List of operation implementations.
-----------------	------------------------------------

tccq_op_registry_add	<i>Add operation implementations to a registry</i>
----------------------	--

Description

Add operation implementations to a registry

Usage

tccq_op_registry_add(registry, implementations)

Arguments

- registry Operation registry.
- implementations Operation implementation or list of implementations.

tccq_op_render	<i>Render an operation implementation for source output</i>
----------------	---

Description

Render an operation implementation for source output

Usage

tccq_op_render(impl, operands, context)

Arguments

- impl Operation implementation.
- operands Rendered operand strings.
- context Rendering context.

tccq_op_render_context

Construct an operation source-rendering context

Description

Construct an operation source-rendering context

Usage

```
tccq_op_render_context(language, backend_id, attrs = list())
```

Arguments

language	Target source language.
backend_id	Backend requesting rendering.
attrs	Structured rendering metadata.

tccq_op_signature

Construct operation signature metadata

Description

Construct operation signature metadata

Usage

```
tccq_op_signature(
  name,
  arity,
  result_type,
  attrs = list(),
  domain_policy = NULL
)
```

Arguments

name	Human-readable operation signature name.
arity	Accepted argument counts or a TccqArity contract.
result_type	Function from input TccqType list to result TccqType.
attrs	Structured signature metadata.
domain_policy	Optional result-shape policy.

tccq_op_signature_result_type

Return an operation signature result type

Description

Return an operation signature result type

Usage

```
tccq_op_signature_result_type(signature, input_types)
```

Arguments

signature	Operation signature.
input_types	List of input TccqType values.

tccq_op_supports

Query operation implementation support

Description

Query operation implementation support

Usage

```
tccq_op_supports(impl, call, context)
```

Arguments

impl	Operation implementation.
call	Observed R call.
context	Operation query context.

tccq_opaque_op_impl

Opaque operation implementation

Description

In R, every call is an operation candidate. This descriptor represents calls whose concrete implementation, purity, and effects have not been refined yet. It is not R call evaluation; that is only one possible backend implementation family.

Usage

```
tccq_opaque_op_impl()
```

tccq_optimize	<i>Optimize a typed program</i>
---------------	---------------------------------

Description

Optimize a typed program

Usage

```
tccq_optimize(optimization, program)
```

Arguments

- optimization Optimization implementation.
- program Typed program to transform.

Value

A [TccqResult](#) containing the transformed [TccqProgram](#).

tccq_plan_backend	<i>Plan a backend for a typed program</i>
-------------------	---

Description

Plan a backend for a typed program

Usage

```
tccq_plan_backend(  
  program,  
  backend = tccq_c_backend(),  
  context = tccq_backend_context()  
)
```

Arguments

- program Program to plan.
- backend Backend implementation.
- context Backend planning context.

tccq_plan_backends	<i>Plan several backends for a typed program</i>
--------------------	--

Description

Every backend accounts for the program and reports constraints through its own plan. The plan set succeeds when at least one backend produces a working plan; backends that cannot lower the program contribute feasibility diagnostics without vetoing the suite.

Usage

```
tccq_plan_backends(
  program,
  backends = tccq_core_backends(),
  context = tccq_backend_context()
)
```

Arguments

program	Program to plan.
backends	Backend implementation descriptors.
context	Backend planning context.

tccq_program	<i>Construct a program schema</i>
--------------	-----------------------------------

Description

Construct a program schema

Usage

```
tccq_program(
  name,
  formals,
  schedule = NULL,
  values = list(),
  regions = list(),
  result = NULL,
  diagnostics = list(),
  call_index = NULL,
  storage_plan = NULL,
  attrs = list()
)
```

Arguments

name	Program name.
formals	Named list of formal bindings.
schedule	Optional ordered program evaluation schedule.
values	List of IR values.
regions	List of executable code regions.
result	Result value id or object.
diagnostics	List of diagnostics attached to the program.
call_index	Typed frontend call handoff.
storage_plan	Optional middle-end storage plan.
attrs	Structured program attributes.

tccq_program_loop_nest

Plan the loop nest for a single-nest lowered program

Description

Thin wrapper over `tccq_program_loop_nests()` for programs that plan to exactly one nest. Multi-nest programs return a structured diagnostic pointing at the plural planner.

Usage

```
tccq_program_loop_nest(program)
```

Arguments

program	Lowered program.
---------	------------------

tccq_program_loop_nests

Plan the ordered loop nests for a lowered program

Description

This is the pass that turns a lowered value graph into the ordered sequence of loop nests consumed by source printers. Every non-root scalar reduction becomes its own all-reduce nest whose result is a named scalar intermediate, and the program result becomes the final nest, whose body references those intermediates through scalar accesses. Elementwise maps become all-map nests, full reductions become all-reduce nests, per-axis reductions and contractions become mixed nests, and slice values disappear into affine accesses. Programs the planner cannot express honestly return structured diagnostics.

Usage

tccq_program_loop_nests(program)

Arguments

program Lowered program.

tccq_program_schedule *Construct an ordered program evaluation schedule*

Description

values is the lowered value graph used to verify references, binding types, dominance, complete step effects, and structured-body cell ownership. It is checked at this boundary but is not duplicated inside the schedule.

Usage

tccq_program_schedule(steps, result, values, body = NULL)

Arguments

steps Ordered TccqEvaluationStep values.
result Result value id produced by the final step.
values Lowered values referenced by the schedule.
body Optional structured value-producing body.

tccq_reduction_identity
 Return a reducer identity literal

Description

Return a reducer identity literal

Usage

tccq_reduction_identity(spec, type)

Arguments

spec Reduction metadata.
type Result type for the reduction.

tccq_reduction_plan	<i>Build a neutral reduction plan</i>
---------------------	---------------------------------------

Description

Build a neutral reduction plan

Usage

tccq_reduction_plan(spec, value, axes, result_type, id, registry)

Arguments

spec	Reduction metadata.
value	Scalar input-element expression.
axes	Ordered reduce axes.
result_type	Scalar result type.
id	Stable loop-nest id.
registry	Operation registry used to build transition expressions.

tccq_reduction_spec	<i>Construct reduction implementation metadata</i>
---------------------	--

Description

Construct reduction implementation metadata

Usage

```
tccq_reduction_spec(  
  name,  
  identity,  
  combine_op,  
  finalize_op = "",  
  associative = TRUE,  
  commutative = TRUE,  
  attrs = list(),  
  signature = NULL  
)
```

Arguments

name	Human-readable reduction name.
identity	Function from result TccqType to identity TccqLiteral.
combine_op	Registered operation combining the accumulator and current value.
finalize_op	Optional registered operation applied to the completed accumulator and reduced element count.
associative	Whether the reducer is associative.
commutative	Whether the reducer is commutative.
attrs	Structured reduction metadata.
signature	Shared operation signature. By default, reductions accept one input and return a scalar of the input base type.

tccq_reduction_state_component	<i>Construct a reduction-state component</i>
--------------------------------	--

Description

Construct a reduction-state component

Usage

tccq_reduction_state_component(name, target, identity)

Arguments

name	Semantic component name.
target	Neutral local target that stores the component.
identity	Initial scalar literal.

tccq_region	<i>Construct an executable code region</i>
-------------	--

Description

Construct an executable code region

Usage

```
tccq_region(
  id,
  kind,
  values = list(),
  fusion_groups = list(),
  effect = tccq_effect(),
  memory_space = "host",
  touches_rapi = FALSE,
  attrs = list()
)
```

Arguments

id	Stable region id.
kind	Region kind: host, kernel, parallel, or device.
values	List of IR values in the region.
fusion_groups	List of fusion groups in the region.
effect	Region effect summary.
memory_space	Dominant memory space: r, host, device, or opaque.
touches_rapi	Whether the region may call into the R C API.
attrs	Structured region attributes.

tccq_registry_supports

Query whether any registry implementation supports a call

Description

Query whether any registry implementation supports a call

Usage

```
tccq_registry_supports(registry, call, context = tccq_op_context())
```

Arguments

registry	Operation registry.
call	Observed R call.
context	Operation query context.

tccq_repeat	<i>Construct a neutral repeat statement</i>
-------------	---

Description

Construct a neutral repeat statement

Usage

```
tccq_repeat(id, body, semantics)
```

Arguments

id	Stable statement id.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.

tccq_resolve_call	<i>Resolve a call to one operation implementation</i>
-------------------	---

Description

Resolve a call to one operation implementation

Usage

```
tccq_resolve_call(registry, call, context = tccq_op_context())
```

Arguments

registry	Operation registry.
call	Observed R call.
context	Operation query context.

tccq_resolved_op	<i>Construct a resolved operation</i>
------------------	---------------------------------------

Description

Construct a resolved operation

Usage

```
tccq_resolved_op(call, implementation, attrs = list())
```

Arguments

call	Observed R call.
implementation	Selected operation implementation.
attrs	Structured resolution metadata.

tccq_result	<i>Create a compiler result</i>
-------------	---------------------------------

Description

Create a compiler result

Usage

```
tccq_result(success, value = NULL, diagnostics = list())
```

Arguments

success	Whether the phase completed without diagnostics.
value	Result value for the phase.
diagnostics	List of TccqDiagnostic values.

tccq_rtinycc_backend	<i>Rtinycc backend descriptor</i>
----------------------	-----------------------------------

Description

This descriptor models the available TinyCC-backed C path without making it the compiler architecture. It consumes the same typed function interface, neutral body or loop nests, storage plan, and C source artifact as the generic C path, then produces a JIT callable through Rtinycc.

Usage

```
tccq_rtinycc_backend()
```

tccq_shape	<i>Construct a shape</i>
------------	--------------------------

Description

Construct a shape

Usage

tccq_shape(dims = list())

Arguments

dims A list, character vector, integer vector, or single dimension.

tccq_storage_allocation	<i>Construct a physical storage allocation</i>
-------------------------	--

Description

Construct a physical storage allocation

Usage

tccq_storage_allocation(id, type, memory_space = "host")

Arguments

id Stable allocation id.
type Semantic storage type shared by every assigned slot.
memory_space Memory space in which the allocation exists.

tccq_storage_lifetime	<i>Construct a storage lifetime interval</i>
-----------------------	--

Description

Construct a storage lifetime interval

Usage

tccq_storage_lifetime(value_id, defined_at, last_used_at)

Arguments

- | | |
|--------------|---|
| value_id | Value whose lifetime is described. |
| defined_at | Linear definition position in schedule-derived value order. |
| last_used_at | Last position where the value must remain live. |

tccq_storage_plan	<i>Construct a storage plan</i>
-------------------	---------------------------------

Description

Construct a storage plan

Usage

tccq_storage_plan(slots = list())

Arguments

- | | |
|-------|------------------------|
| slots | List of storage slots. |
|-------|------------------------|

tccq_storage_slot	<i>Construct a storage slot</i>
-------------------	---------------------------------

Description

Construct a storage slot

Usage

```
tccq_storage_slot(  
    id,  
    value_id,  
    type,  
    role,  
    materialized,  
    lifetime = NULL,  
    allocation = NULL  
)
```

Arguments

id	Stable storage-slot id.
value_id	Value owned by this slot.
type	Slot value type.
role	Storage role.
materialized	Whether storage exists at runtime.
lifetime	Optional typed liveness interval for storage reuse.
allocation	Physical allocation for a materialized temporary.

tccq_subscript_spec	<i>Construct subscript implementation metadata</i>
---------------------	--

Description

Construct subscript implementation metadata

Usage

```
tccq_subscript_spec(name, signature)
```

Arguments

name	Human-readable subscript operation name.
signature	Shared operation signature.

tccq_switch	<i>Construct a neutral positional switch</i>
-------------	--

Description

Construct a neutral positional switch

Usage

tccq_switch(id, selector, selector_target, alternatives, semantics)

Arguments

- | | |
|-----------------|--|
| id | Stable statement id. |
| selector | Scalar integer selector expression. |
| selector_target | Typed local target for the evaluate-once selector. |
| alternatives | Ordered positional alternative blocks. |
| semantics | Evaluator facts for the originating switch special form. |

tccq_type	<i>Construct a compiler type</i>
-----------	----------------------------------

Description

Construct a compiler type

Usage

tccq_type(base, shape = tccq_shape())

Arguments

- | | |
|-------|------------------------------|
| base | Scalar or storage base type. |
| shape | Shape value. |

tccq_unimplemented_calls	<i>Find calls without an implementation for a registry and context</i>
--------------------------	--

Description

Find calls without an implementation for a registry and context

Usage

```
tccq_unimplemented_calls(  
    calls,  
    registry = tccq_default_op_registry(),  
    context = tccq_op_context()  
)
```

Arguments

- | | |
|----------|---------------------------|
| calls | List of TccqCall objects. |
| registry | Operation registry. |
| context | Operation query context. |

tccq_value	<i>Construct an IR value</i>
------------	------------------------------

Description

Construct an IR value

Usage

```
tccq_value(  
    id,  
    op,  
    inputs = list(),  
    type,  
    effect = tccq_effect(),  
    attrs = list()  
)
```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.
effect	Effect summary.
attrs	Structured operation attributes.

tccq_value_block	<i>Construct a value-producing neutral statement block</i>
------------------	--

Description

Construct a value-producing neutral statement block

Usage

```
tccq_value_block(id, result, locals = list(), statements = list())
```

Arguments

id	Stable block id.
result	Typed target produced on every terminal path through the block.
locals	Ordered local write targets declared by the block.
statements	Ordered TccqStatement values.

tccq_verify_optimization	<i>Verify an optimized program</i>
--------------------------	------------------------------------

Description

Verify an optimized program

Usage

```
tccq_verify_optimization(optimization, before, after)
```

Arguments

optimization	Optimization implementation.
before	Program before transformation.
after	Program after transformation.

Value

A [TccqResult](#) containing after when its pass invariants hold.

tccq_while	<i>Construct a neutral while statement</i>
------------	--

Description

Construct a neutral while statement

Usage

```
tccq_while(id, condition, body, semantics)
```

Arguments

- | | |
|-----------|--|
| id | Stable statement id. |
| condition | Scalar logical condition expression. |
| body | Procedural loop body. |
| semantics | Evaluator facts for the originating R loop special form. |

tccq_write_target	<i>Construct a neutral write target</i>
-------------------	---

Description

Construct a neutral write target

Usage

```
tccq_write_target(  
  value_id,  
  type,  
  storage_type = tccq_type(type@base),  
  kind = "local",  
  binding = NULL  
)
```

Arguments

- | | |
|--------------|--|
| value_id | Stable value id written by the target. |
| type | Semantic value type. |
| storage_type | Scalar storage type written at the target. |
| kind | Target kind: local, result, or cell. |
| binding | Optional binding owned by the target. |

TccqAccess	<i>Domain access mapping</i>
------------	------------------------------

Description

Access describes how a value is read or written over a domain. It is the schema-level place for identity indexing, scalar broadcast, slicing, transpose, and later richer affine/custom maps.

Usage

```
TccqAccess(
  value_id = character(0),
  domain = TccqDomain(),
  kind = character(0),
  index_map = list(),
  consumer_shape = NULL
)
```

Arguments

value_id	Referenced value id.
domain	Access domain.
kind	Access kind.
index_map	Structured index-map payload.
consumer_shape	Logical consumer shape for recycling, or NULL for other access kinds.

TccqArgReductionSpec	<i>Argument-selection reduction metadata</i>
----------------------	--

Description

An argument reduction tracks whether a value was seen, the selected value, and its one-based index. The first concrete form supports R-compatible maximum selection while ignoring missing double values.

Usage

```
TccqArgReductionSpec(
  name = character(0),
  signature = TccqOpSignature(),
  associative = logical(0),
  commutative = logical(0),
  empty_policy = character(0),
  attrs = list(),
```

```
direction = character(0),
missing = character(0),
ties = character(0)
)
```

Arguments

name	Human-readable reduction name.
signature	Shared operation signature.
associative	Whether the reducer is associative under its declared semantics.
commutative	Whether the reducer is commutative under its declared semantics.
empty_policy	Behavior when no value contributes to the reduction.
attrs	Structured reduction metadata.
direction	Value-selection direction.
missing	Missing-value policy.
ties	Tie-selection policy.

TccqArity	<i>Operation arity contract</i>
-----------	---------------------------------

Description

Arity is a typed operation fact rather than an expanded vector of convenient argument counts. Exact contracts enumerate accepted counts; interval contracts own inclusive lower and optional upper bounds. This lets rank-polymorphic operations such as [remain independent of an arbitrary maximum array rank.

Usage

```
TccqArity(counts = integer(0), minimum = NULL, maximum = NULL)
```

Arguments

counts	Exact accepted argument counts.
minimum	Inclusive lower argument-count bound, or NULL for an exact contract.
maximum	Inclusive upper argument-count bound, or NULL for no upper bound.

TccqAssignment	<i>Neutral assignment statement</i>
----------------	-------------------------------------

Description

Neutral assignment statement

Usage

```
TccqAssignment(  
  id = character(0),  
  effect = TccqEffect(),  
  target = TccqWriteTarget(),  
  value = TccqExpression()  
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
target	Typed destination.
value	Backend-neutral value expression.

TccqBackend	<i>Backend implementation trait</i>
-------------	-------------------------------------

Description

Backend implementations must opt into this trait before compiler code uses them for planning.

Usage

```
TccqBackend
```

TccqBackendAllocationBinding

Backend allocation binding

Description

An allocation binding assigns one generated source identifier to one typed physical allocation. Every materialized slot sharing that allocation is carried by the same binding.

Usage

```
TccqBackendAllocationBinding(
  source_name = character(0),
  allocation = TccqStorageAllocation(),
  slots = list()
)
```

Arguments

source_name	Generated source identifier.
allocation	Physical allocation represented by the identifier.
slots	Materialized temporary slots sharing the allocation.

TccqBackendArtifact *Backend artifact*

Description

Backend artifacts are concrete products of a backend plan: generated source, a compiled shared library, or a JIT callable. They are explicit plan values so source, shared-library, and JIT modes cannot silently collapse into the same loose string attribute.

Usage

```
TccqBackendArtifact(
  id = character(0),
  kind = character(0),
  source_language = character(0),
  path = character(0),
  attrs = list()
)
```

Arguments

id	Stable artifact id.
kind	Artifact kind.
source_language	Source language used to produce the artifact, or empty.
path	Filesystem path for file-backed artifacts, or empty.
attrs	Structured artifact metadata.

TccqBackendContext	<i>Backend planning context</i>
--------------------	---------------------------------

Description

Backend planning context

Usage

```
TccqBackendContext(
  mode = character(0),
  target = character(0),
  allow_boundary = logical(0),
  required_capabilities = character(0),
  attrs = list()
)
```

Arguments

mode	Requested backend mode.
target	Requested target language or runtime, or any.
allow_boundary	Whether explicit R call-evaluation boundary plans are allowed.
required_capabilities	Capabilities that candidate backends must expose.
attrs	Structured backend-context attributes.

TccqBackendErrorChannel

Backend callable error channel

Description

Generated code writes zero on success or the one-based position of a diagnostic on failure. Wrappers turn the status back into the corresponding classed runtime diagnostic.

Usage

```
TccqBackendErrorChannel(source_name = character(0), diagnostics = list())
```

Arguments

source_name	Generated status-output parameter name.
diagnostics	Ordered runtime diagnostics addressable by status code.

TccqBackendExtentBinding

Backend extent binding

Description

An extent binding assigns one generated ABI parameter to one declared dimension symbol.

Usage

```
TccqBackendExtentBinding(source_name = character(0), symbol = character(0))
```

Arguments

source_name	Generated extent parameter name.
symbol	Declared dimension symbol.

TccqBackendFunctionInterface

Backend function interface

Description

A backend function interface is the source-level callable boundary consumed by C, Rtinycc, Fortran, and later source printers. Generated parameters, scalar locals, physical allocations, and extents are closed typed bindings; positional arrays are not an interface protocol.

Usage

```
TccqBackendFunctionInterface(
    symbol = character(0),
    source_language = character(0),
    kind = character(0),
    abi = character(0),
    parameters = list(),
    locals = list(),
    allocations = list(),
    result_value_id = character(0),
    result_type = TccqType(),
    result_placement = character(0),
    result_name = character(0),
    domain = NULL,
    extents = list(),
    index_names = character(0),
    result_dims = list(),
    result_count_name = character(0),
    error_channel = NULL
)
```

Arguments

symbol	Generated function symbol.
source_language	Source language consumed by the printer.
kind	Function shape: scalar, loop_nest, or structured.
abi	Callable ABI.
parameters	Ordered ABI parameter bindings.
locals	Ordered scalar-local bindings.
allocations	Ordered physical-allocation bindings.
result_value_id	Lowered result value id.
result_type	Declared semantic result type.

result_placement	Whether the result is returned or passed by output argument.
result_name	Generated result/output variable name, or empty string.
domain	Loop-nest iteration domain, or NULL.
extents	Ordered symbolic-extent bindings carried by the ABI.
index_names	Generated data-parallel and structured loop index names.
result_dims	Typed result dimensions.
result_count_name	Generated result element-count parameter name, or empty string for scalar results.
error_channel	Optional generated callable error channel.

TccqBackendPlan	<i>Backend plan</i>
-----------------	---------------------

Description

Backend plan

Usage

```
TccqBackendPlan(
    id = character(0),
    backend_id = character(0),
    family = character(0),
    mode = character(0),
    target = character(0),
    capabilities = character(0),
    regions = list(),
    bridges = list(),
    diagnostics = list(),
    products = NULL,
    attrs = list()
)
```

Arguments

id	Stable plan id.
backend_id	Backend spec id.
family	Backend family.
mode	Backend mode.
target	Target language or runtime.
capabilities	Backend capabilities visible to this plan.

regions	Regions assigned to the backend plan.
bridges	Representation-transition plans.
diagnostics	Diagnostics attached to this plan.
products	Typed backend products.
attrs	Structured plan attributes.

TccqBackendPlanSet	<i>Backend plan set</i>
--------------------	-------------------------

Description

A plan set is the neutral result of asking several backend families to account for the same typed program. It exists to keep the core honest: C, Fortran, and future graph, device, or R call-evaluation implementations must report their constraints through the same contract.

Usage

```
TccqBackendPlanSet(
  program_name = character(0),
  plans = list(),
  diagnostics = list(),
  attrs = list()
)
```

Arguments

program_name	Program name.
plans	Backend plans.
diagnostics	Diagnostics collected while planning.
attrs	Structured plan-set attributes.

TccqBackendProducts	<i>Backend products</i>
---------------------	-------------------------

Description

Backend products are the typed outputs owned by a backend plan: the callable interface, loop nest, expression or statement block consumed by the source printer, storage plan, and concrete artifacts. Runtime handles such as loaded libraries and compiled callables stay in `attrs`, but under this typed `products` value rather than directly on the backend plan.

Usage

```
TccqBackendProducts(  
  function_interface = NULL,  
  body = NULL,  
  loop_nest = NULL,  
  loop_nests = list(),  
  storage_plan = NULL,  
  artifacts = list(),  
  attrs = list()  
)
```

Arguments

function_interface	Source-level callable interface, if source exists.
body	Expression tree or statement block consumed by the backend, if source exists.
loop_nest	Result loop nest consumed by the backend, if source exists.
loop_nests	Ordered loop nests, intermediates first, result nest last.
storage_plan	Storage plan consumed by the backend, if available.
artifacts	Named backend artifacts.
attrs	Structured product metadata.

TccqBackendSpec	<i>Backend implementation descriptor</i>
-----------------	--

Description

Backend implementation descriptor

Usage

```
TccqBackendSpec(  
  id = character(0),  
  family = character(0),  
  target = character(0),  
  driver = character(0),  
  modes = character(0),  
  region_kinds = character(0),  
  memory_spaces = character(0),  
  capabilities = character(0),  
  uses_rapi = logical(0),  
  attrs = list(),  
  prepare = function() NULL  
)
```

Arguments

id	Stable backend id.
family	Backend family.
target	Target language or runtime.
driver	Concrete driver name.
modes	Supported backend modes.
region_kinds	Supported region kinds.
memory_spaces	Supported memory spaces.
capabilities	Capabilities exposed by this backend.
uses_rapi	Whether this backend may touch the R C API.
attrs	Structured backend attributes.
prepare	Planning implementation.

TccqBackendValueBinding

Backend value binding

Description

A value binding assigns one generated source identifier and source type to a neutral value. Parameters and scalar locals share this representation so their names, identities, and types cannot drift in parallel arrays.

Usage

```
TccqBackendValueBinding(
  source_name = character(0),
  value_id = character(0),
  source_type = TccqType(),
  role = character(0)
)
```

Arguments

source_name	Generated source identifier.
value_id	Neutral value id.
source_type	Type represented by the generated identifier.
role	Whether the identifier is an ABI parameter or scalar local.

TccqBinding	<i>Program binding</i>
-------------	------------------------

Description

Program binding

Usage

```
TccqBinding(name = character(0), type = TccqType(), mutable = logical(0))
```

Arguments

name	Binding name.
type	Binding type.
mutable	Whether the binding may be mutated.

TccqBindingReference	<i>Local binding reference</i>
----------------------	--------------------------------

Description

A binding reference is a lexical read of an already evaluated local. It retains the exact binding identity while pointing at the binding's storage value. Expression and effect traversal stop at this node; reading a local must not re-evaluate the graph that defined it.

Usage

```
TccqBindingReference(
  id = character(0),
  op = character(0),
  inputs = list(),
  type = TccqType(),
  effect = TccqEffect(),
  attrs = list(),
  binding = TccqLocalBinding()
)
```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.

effect	Effect summary.
attrs	Structured operation attributes.
binding	Local binding being read.

TccqBlock	<i>Neutral statement block</i>
-----------	--------------------------------

Description

A block owns lexical local declarations and ordered typed statements. Its nesting is semantic: locals and statements inside one branch arm are not evaluated from another arm. A plain block does not imply that evaluation yields a value; [TccqValueBlock](#) adds that stronger completion contract.

Usage

```
TccqBlock(
  id = character(0),
  locals = list(),
  statements = list(),
  effect = TccqEffect()
)
```

Arguments

id	Stable block id.
locals	Ordered local write targets declared by the block.
statements	Ordered TccqStatement values.
effect	Exact effect summary of evaluating the statements.

TccqBranch	<i>Lazy conditional IR value</i>
------------	----------------------------------

Description

TccqBranch is the value-level representation of R's if special form. Its three incoming value ids are analyzed eagerly by the compiler, but the recorded special-form semantics require generated code to evaluate exactly one result arm at runtime.

Usage

```

TccqBranch(
  id = character(0),
  op = character(0),
  inputs = list(),
  type = TccqType(),
  effect = TccqEffect(),
  attrs = list(),
  condition = character(0),
  consequent = character(0),
  alternative = character(0),
  semantics = TccqCallSemantics()
)

```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.
effect	Effect summary.
attrs	Structured operation attributes.
condition	Scalar logical condition value id.
consequent	Value id produced when the condition is true.
alternative	Value id produced when the condition is false.
semantics	Evaluator facts for the originating if call.

TccqBridgePlan

Backend bridge plan

Description

Bridges model representation transitions such as `SEXP -> host buffer`, `host -> device`, layout conversion, tile materialization, or R call-evaluation boundaries. They are explicit backend-plan values, not emitter branches.

Usage

```

TccqBridgePlan(
  id = character(0),
  kind = character(0),
  from_space = character(0),
  to_space = character(0),
  from_type = NULL,

```

```

    to_type = NULL,
    effect = TccqEffect(),
    attrs = list()
)

```

Arguments

id	Stable bridge id.
kind	Bridge kind.
from_space	Source memory space.
to_space	Destination memory space.
from_type	Optional source type.
to_type	Optional destination type.
effect	Bridge effect summary.
attrs	Structured bridge attributes.

TccqCall	<i>R call observed by the frontend</i>
----------	--

Description

R call observed by the frontend

Usage

```

TccqCall(
  id = character(0),
  name = character(0),
  expr = NULL,
  origin = character(0),
  kind = character(0),
  arity = integer(0),
  argument_names = character(0),
  attrs = list()
)

```

Arguments

id	Stable call id.
name	Call name.
expr	Original R call expression.
origin	Source of the call observation.
kind	Structural call kind.
arity	Number of supplied arguments, or NA_integer_.
argument_names	Supplied argument tags.
attrs	Structured call attributes.

TccqCallIndex	<i>Indexed frontend call facts</i>
---------------	------------------------------------

Description

TccqCallIndex is the typed handoff from frontend analysis to later lowering passes. It keeps observed calls and evaluator facts aligned by stable call id.

Usage

```
TccqCallIndex(calls = list(), semantics = list(), attrs = list())
```

Arguments

calls	List of observed calls.
semantics	List of call evaluator facts.
attrs	Structured index attributes.

TccqCallSemantics	<i>R call evaluator facts</i>
-------------------	-------------------------------

Description

TccqCallSemantics records evaluator-level facts about an observed call. It is not a lowering decision: special forms, builtins, closures, replacement calls, group generics, and unknown calls still need later operation and backend implementations.

Usage

```
TccqCallSemantics(
  call = TccqCall(),
  evaluator_kind = character(0),
  forcing_policy = character(0),
  dispatch_kind = character(0),
  lexical_scope = logical(0),
  replacement = logical(0),
  control = logical(0),
  attrs = list()
)
```

Arguments

call	Observed call.
evaluator_kind	Evaluator kind.
forcing_policy	Promise-forcing policy.
dispatch_kind	Dispatch family.
lexical_scope	Whether lexical closure environment matters.
replacement	Whether this is a replacement-function call.
control	Whether this is a control-form call.
attrs	Structured semantic attributes.

TccqCell	<i>Mutable lexical cell</i>
----------	-----------------------------

Description

A cell is storage for loop-carried scalar state. It is deliberately distinct from [TccqLocalBinding](#), whose identity remains one immutable SSA definition. Source names are assigned later by the backend interface.

Usage

```
TccqCell(  
  name = character(0),  
  type = TccqType(),  
  mutable = logical(0),  
  value_id = character(0),  
  storage_type = TccqType()  
)
```

Arguments

name	Binding name.
type	Binding type.
mutable	Whether the binding may be mutated.
value_id	Stable storage identity.
storage_type	Scalar storage type.

TccqCellReference	<i>Lowered mutable-cell read</i>
-------------------	----------------------------------

Description

Lowered mutable-cell read

Usage

```
TccqCellReference(  
  id = character(),  
  op = character(),  
  inputs = list(),  
  type = TccqType(),  
  effect = TccqEffect(),  
  attrs = list(),  
  cell = TccqCell()  
)
```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.
effect	Effect summary.
attrs	Structured operation attributes.
cell	Mutable cell being read.

TccqConditional	<i>Value-producing neutral conditional</i>
-----------------	--

Description

This is the stricter value-producing form of TccqIf. Both arms must be value blocks writing the same target, and the retained [TccqBranch](#) must agree with the procedural control facts. The branch retains the effect of the source value; the statement effect covers only work left in its normalized arms after reductions or contractions have been extracted.

Usage

```
TccqConditional(
  id = character(0),
  effect = TccqEffect(),
  condition = TccqExpression(),
  consequent = TccqBlock(),
  alternative = TccqBlock(),
  semantics = TccqCallSemantics(),
  branch = TccqBranch()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
condition	Scalar logical condition expression.
consequent	Block evaluated when the condition is true.
alternative	Block evaluated when the condition is false.
semantics	Evaluator facts for the originating if special form.
branch	Source branch payload retaining R special-form semantics.

TccqContractionSpec	<i>Contraction implementation metadata</i>
---------------------	--

Description

A contraction spec is attached to an operation implementation when calls to that implementation lower to a loop nest that multiplies (or otherwise combines) aligned elements along shared reduce axes and folds them with a reducer, such as `%*%`. It carries the shared operation signature, the reducer used for the fold, and the elementwise combine operation applied before folding.

Usage

```
TccqContractionSpec(
  name = character(0),
  signature = TccqOpSignature(),
  reducer = TccqFoldReductionSpec(),
  combine_op = character(0),
  attrs = list()
)
```

Arguments

name	Human-readable contraction name.
signature	Shared operation signature.
reducer	Reduction metadata used for the contracted axes.
combine_op	Elementwise operation name combining aligned elements.
attrs	Structured contraction metadata.

TccqControlCompletion *Structured control completion*

Description

Completion is distinct from [TccqEffect](#). It records whether structured control may reach the next statement or transfer to the nearest enclosing loop. These facts let dominance and later transformations reason about normal paths without treating break or next as fabricated side effects.

Usage

```
TccqControlCompletion(
    falls_through = logical(0),
    breaks = logical(0),
    continues = logical(0)
)
```

Arguments

falls_through	Whether evaluation may reach the next statement.
breaks	Whether evaluation may break from the nearest enclosing loop.
continues	Whether evaluation may continue the nearest enclosing loop.

TccqDeadStoreOptimization
 Dead-store optimization

Description

This optimization removes assignments to compiler-owned local or cell storage when the target is never read and evaluating the assigned expression cannot write, allocate, cross a boundary, signal an error, or warn. It is deliberately limited to structured neutral program bodies; graph-level dead value elimination must also rewrite schedules, regions, fusion contracts, storage lifetimes, and allocations and therefore requires a separate pass.

Usage

```
TccqDeadStoreOptimization()
```

TccqDiagnostic	<i>Compiler diagnostic value</i>
----------------	----------------------------------

Description

Diagnostics are values first and condition payloads second. Compiler code should pass these objects around instead of branching on condition strings.

Usage

```
TccqDiagnostic(  
  code = character(0),  
  message = character(0),  
  phase = character(0),  
  path = character(0),  
  data = list()  
)
```

Arguments

code	Stable diagnostic code.
message	Human-facing message.
phase	Compiler phase that produced the diagnostic.
path	Optional path into the program or schema.
data	Structured diagnostic payload.

TccqDim	<i>Shape dimension</i>
---------	------------------------

Description

Shape dimension

Usage

```
TccqDim(kind = character(0), label = character(0), value = integer(0))
```

Arguments

kind	Dimension kind.
label	Symbolic dimension label.
value	Constant dimension value.

TccqDimensionReference

Typed declared-dimension reference

Description

A dimension reference identifies the scalar ABI extent corresponding to one declared symbolic dimension. It is distinct from an ordinary scalar binding with the same source spelling.

Usage

```
TccqDimensionReference(
    source_value_id = character(0),
    symbol = character(0),
    binding = NULL,
    slice_offsets = integer(0),
    access = NULL,
    dimension = TccqDim()
)
```

Arguments

source_value_id	Logical value id supplying the referenced storage.
symbol	Optional source symbol, or an empty string when not applicable.
binding	Optional lexical binding represented by the reference.
slice_offsets	Zero-based slice offsets, or an empty integer vector.
access	Optional typed domain access attached by loop-nest planning.
dimension	Declared symbolic dimension supplied by the reference.

TccqDomain

Iteration domain

Description

A domain is the semantic iteration space for elementwise, reduction, stencil, or tiled work. It is separate from shape so transformed programs can name and reuse iteration spaces directly.

Usage

```
TccqDomain(
    id = character(0),
    shape = TccqShape(),
    axes = character(0),
    attrs = list()
)
```

Arguments

id	Stable domain id.
shape	Domain shape.
axes	Axis names.
attrs	Structured domain attributes.

TccqDomainPolicy	<i>Operation domain policy</i>
------------------	--------------------------------

Description

A domain policy describes how input shapes determine an operation result shape before target source generation. It is the typed home for rules such as scalar broadcast, common elementwise domains, and scalar reducer results.

Usage

```
TccqDomainPolicy(
    name = character(0),
    result_shape = function() NULL,
    attrs = list()
)
```

Arguments

name	Human-readable domain policy name.
result_shape	Function from input TccqType list to result TccqShape.
attrs	Structured domain-policy metadata.

TccqEffect	<i>Compiler effect summary</i>
------------	--------------------------------

Description

Compiler effect summary

Usage

```
TccqEffect(
    reads = logical(0),
    writes = logical(0),
    allocates = logical(0),
    boundary = logical(0),
    may_error = logical(0),
    may_warn = logical(0)
)
```

Arguments

reads	Whether the operation reads program state.
writes	Whether the operation writes program state.
allocates	Whether the operation allocates storage.
boundary	Whether the operation crosses an unsupported boundary.
may_error	Whether the operation may signal at runtime.
may_warn	Whether the operation may warn at runtime.

TccqElementwiseSpec	<i>Elementwise implementation metadata</i>
---------------------	--

Description

An elementwise spec is attached to an operation implementation when calls to that implementation can lower to elementwise values over the compiler domain model. It carries an operation signature so lowering does not recognize elementwise operations by spelling.

Usage

```
TccqElementwiseSpec(
  name = character(0),
  signature = TccqOpSignature(),
  attrs = list()
)
```

Arguments

name	Human-readable elementwise operation name.
signature	Shared operation signature.
attrs	Structured elementwise metadata.

TccqEvaluationStep	<i>Scheduled R evaluation</i>
--------------------	-------------------------------

Description

One step records one executable top-level form after declarations have been removed. Assignment steps carry the local binding they define; expression statements carry no binding. The effect summarizes the complete value graph evaluated by the step.

Usage

```

TccqEvaluationStep(
    value_id = character(0),
    statement_index = integer(0),
    binding = NULL,
    uses = list(),
    effect = TccqEffect()
)

```

Arguments

value_id	Lowered value evaluated by this step.
statement_index	One-based executable statement position.
binding	Optional local binding defined by the evaluation.
uses	Local bindings read by the evaluation.
effect	Effect of evaluating the complete step.

TccqExpression	<i>Backend-neutral expression tree</i>
----------------	--

Description

TccqExpression is the small tree form consumed by source printers. It is built from typed lowered values and resolved operations, so C, Fortran, TinyCC, and later printers do not rediscover expression semantics from raw operation strings.

Usage

```

TccqExpression(
    id = character(0),
    kind = character(0),
    value_id = character(0),
    op = character(0),
    inputs = list(),
    type = TccqType(),
    effect = TccqEffect(),
    literal = NULL,
    operation = NULL,
    branch = NULL,
    reference = NULL,
    index_proofs = list()
)

```

Arguments

id	Stable expression id.
kind	Expression kind.
value_id	Lowered value id represented by this expression.
op	Operation name, or formal/literal for leaves.
inputs	Child expressions.
type	Result type.
effect	Effect of evaluating the expression.
literal	Literal payload for literal expressions.
operation	Typed lowered-operation payload for operation expressions.
branch	Typed conditional payload for branch expressions.
reference	Typed source payload for reference expressions.
index_proofs	Per-axis bounds proofs for indexed expressions.

TccqExpressionReference

Typed expression reference

Description

A reference names the logical source value read by a neutral expression. It keeps lexical binding, source symbol, slice, and affine access facts out of open-ended expression metadata. Physical allocation remains a separate storage-plan concern.

Usage

```
TccqExpressionReference(
  source_value_id = character(0),
  symbol = character(0),
  binding = NULL,
  slice_offsets = integer(0),
  access = NULL
)
```

Arguments

source_value_id	Logical value id supplying the referenced storage.
symbol	Optional source symbol, or an empty string when not applicable.
binding	Optional lexical binding represented by the reference.
slice_offsets	Zero-based slice offsets, or an empty integer vector.
access	Optional typed domain access attached by loop-nest planning.

TccqFoldReductionSpec *Scalar fold reduction metadata*

Description

A fold reduction has one accumulator identity, one combine operation, and an optional finalizer. Its neutral loop state has exactly one component.

Usage

```
TccqFoldReductionSpec(
  name = character(0),
  signature = TccqOpSignature(),
  associative = logical(0),
  commutative = logical(0),
  empty_policy = character(0),
  attrs = list(),
  identity = function() NULL,
  combine_op = character(0),
  finalize_op = character(0)
)
```

Arguments

name	Human-readable reduction name.
signature	Shared operation signature.
associative	Whether the reducer is associative under its declared semantics.
commutative	Whether the reducer is commutative under its declared semantics.
empty_policy	Behavior when no value contributes to the reduction.
attrs	Structured reduction metadata.
identity	Function from result <code>TccqType</code> to identity <code>TccqLiteral</code> .
combine_op	Registered elementwise operation combining the accumulator and current value.
finalize_op	Optional registered elementwise operation combining the completed accumulator and reduced element count.

TccqFor *Neutral for statement*

Description

A for loop evaluates one typed iteration plan and assigns each selected element to its iteration cell before evaluating the body. Source evaluation, element selection, and iteration domain are explicit so source backends do not reconstruct traversal from R syntax.

Usage

```
TccqFor(
  id = character(0),
  effect = TccqEffect(),
  body = TccqBlock(),
  semantics = TccqCallSemantics(),
  iterator = TccqWriteTarget(),
  iteration = TccqIterationPlan()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.
iterator	Mutable scalar destination receiving each element.
iteration	Backend-neutral iteration plan.

TccqFusionContract	<i>Fusion operation contract</i>
--------------------	----------------------------------

Description

TccqFusionContract is the typed payload attached to a fusion group after region planning. It keeps the lowered operation payloads, signatures, domain policies, typed result value, optional result operation, and storage strategy together so fusion legality and later optimization passes do not inspect loose group attributes. Control values such as TccqBranch are valid fusion results even when the group contains no lowered operations.

Usage

```
TccqFusionContract(
  fusion_kind = character(0),
  storage_strategy = character(0),
  operations = list(),
  result_value = TccqValue(),
  result_operation = NULL,
  operation_signatures = list(),
  domain_policies = list(),
  attrs = list()
)
```

Arguments

fusion_kind	Fusion kind owned by the contract.
storage_strategy	Storage strategy requested by the fusion plan.
operations	Named lowered operations by value id.
result_value	Typed value produced by the fusion group.
result_operation	Optional lowered operation carried by result_value.
operation_signatures	Named operation signatures by value id.
domain_policies	Named domain policies by value id.
attrs	Structured metadata.

TccqFusionGroup	<i>Fusion group</i>
-----------------	---------------------

Description

A fusion group is a typed plan candidate, not emitted code. It groups values that may share one domain and region when operation implementations, effects, access maps, layout, and backend constraints allow it.

Usage

```
TccqFusionGroup(
    id = character(0),
    kind = character(0),
    domain = TccqDomain(),
    values = list(),
    outputs = character(0),
    accesses = list(),
    region_kind = character(0),
    target = character(0),
    effect = TccqEffect(),
    contract = NULL,
    attrs = list()
)
```

Arguments

id	Stable fusion-group id.
kind	Fusion kind.
domain	Shared iteration domain.

values	List of IR values in the group.
outputs	Output value ids.
accesses	List of access mappings.
region_kind	Candidate execution region kind, or any.
target	Candidate implementation target, or any.
effect	Effect summary for the fused group.
contract	Typed fusion operation/storage contract.
attrs	Structured fusion attributes.

TccqIf	<i>Neutral procedural if statement</i>
--------	--

Description

Neutral procedural if statement

Usage

```
TccqIf(  
  id = character(0),  
  effect = TccqEffect(),  
  condition = TccqExpression(),  
  consequent = TccqBlock(),  
  alternative = TccqBlock(),  
  semantics = TccqCallSemantics()  
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
condition	Scalar logical condition expression.
consequent	Block evaluated when the condition is true.
alternative	Block evaluated when the condition is false.
semantics	Evaluator facts for the originating if special form.

TccqIndexedValue	<i>Proven scalar indexed read</i>
------------------	-----------------------------------

Description

This value represents one scalar extraction only when every source axis has a [TccqIndexProof](#). Proofs may share an iteration, as in a diagonal read. The stored access is zero-based; iterator cells retain R's one-based values. No general R subscript semantics are implied by this class.

Usage

```
TccqIndexedValue(
  id = character(),
  op = character(),
  inputs = list(),
  type = TccqType(),
  effect = TccqEffect(),
  attrs = list(),
  source_type = TccqType(),
  index_proofs = list(),
  access = TccqAccess(),
  operation = TccqLoweredOperation(),
  semantics = TccqCallSemantics()
)
```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.
effect	Effect summary.
attrs	Structured operation attributes.
source_type	Type of the source storage.
index_proofs	Per-axis affine selector and bounds proofs.
access	Zero-based source access corresponding to the proofs.
operation	Lowered subscript implementation payload.
semantics	Evaluator facts for the originating [call.

TccqIndexExpr	<i>Affine index expression</i>
---------------	--------------------------------

Description

An index expression maps one stored-tensor axis into the iteration space of a loop nest as `axis + offset`, or to a constant position when `axis` is empty. It is the typed unit of access maps, so stencil shifts, broadcasts, and transposes are dimension facts rather than printed index strings.

Usage

```
TccqIndexExpr(axis = character(0), offset = integer(0))
```

Arguments

<code>axis</code>	Iteration-axis name, or empty for a constant index.
<code>offset</code>	Signed integer offset (zero-based).

TccqIndexProof	<i>Proven affine scalar index</i>
----------------	-----------------------------------

Description

An index proof ties one R iterator cell to one source axis. The iteration establishes the cell range, `index` is the zero-based storage access, and `source_extent` proves that the full affine access remains in bounds. A non-zero offset additionally retains the resolved `+` call and its evaluator facts; the range proof is what discharges integer overflow and subscript warnings for that normalized selector.

Usage

```
TccqIndexProof(
  iterator = TccqWriteTarget(),
  selector = TccqCellReference(),
  iteration = TccqIterationPlan(),
  source_extent = TccqDim(),
  index = TccqIndexExpr(),
  operation = NULL,
  semantics = NULL
)
```

Arguments

iterator	Iterator target populated by the enclosing virtual loop.
selector	Direct reference to the iterator cell.
iteration	Virtual iteration establishing the selector range.
source_extent	Extent of the indexed source axis.
index	Zero-based affine storage index.
operation	Lowered selector operation for a shifted index, or NULL.
semantics	Evaluator facts for the shifted selector call, or NULL.

TccqIterationPlan	<i>Backend-neutral iteration plan</i>
-------------------	---------------------------------------

Description

An iteration plan separates one-time source evaluation from selection of the current element. Stored-vector iteration carries a typed element expression with an access map. Virtual unit sequences carry an affine index expression and the resolved operation implementation that defines the sequence.

Usage

```
TccqIterationPlan(
    source = TccqExpression(),
    domain = TccqDomain(),
    element = TccqExpression(),
    element_type = TccqType(),
    resolved_op = NULL,
    semantics = NULL,
    effect = TccqEffect()
)
```

Arguments

source	Expression evaluated once to establish iteration.
domain	One-dimensional iteration domain.
element	Current element expression or affine induction value.
element_type	Scalar type assigned to the iteration cell.
resolved_op	Resolved operation for virtual iteration, or NULL for stored-value iteration.
semantics	Evaluator facts for the virtual iteration call, or NULL.
effect	Effect summary for source evaluation and element selection.

TccqIterationSpec	<i>Iteration implementation metadata</i>
-------------------	--

Description

An iteration spec describes an operation whose result can be consumed as a virtual one-dimensional iteration space. The shared signature owns argument and result typing; extent_arg selects the argument proving the domain extent, while start defines the first affine induction value. The current form deliberately represents unit-stride sequences only.

Usage

```
TccqIterationSpec(  
    name = character(0),  
    signature = TccqOpSignature(),  
    extent_arg = integer(0),  
    start = integer(0)  
)
```

Arguments

name	Human-readable iteration operation name.
signature	Shared operation signature.
extent_arg	One-based argument position supplying the domain extent.
start	First integer value produced by the iteration.

TccqLiteral	<i>Scalar literal value</i>
-------------	-----------------------------

Description

Scalar literal value

Usage

```
TccqLiteral(kind = character(0), type = TccqType(), value = NULL)
```

Arguments

kind	Literal kind.
type	Scalar type.
value	Literal payload.

TccqLocalBinding	<i>Local value binding</i>
------------------	----------------------------

Description

A local binding records the value produced by one top-level assignment in the declared subset. Its statement position is a semantic definition boundary used by scheduling; it is not source-printer metadata.

Usage

```
TccqLocalBinding(
  name = character(0),
  type = TccqType(),
  mutable = logical(0),
  value_id = character(0),
  statement_index = integer(0)
)
```

Arguments

name	Binding name.
type	Binding type.
mutable	Whether the binding may be mutated.
value_id	Lowered value bound to the local name.
statement_index	One-based executable statement position defining the binding.

TccqLoop	<i>Neutral sequential loop</i>
----------	--------------------------------

Description

Abstract parent for sequential loops whose body is a typed statement block. Concrete loop classes add their own entry and completion rules.

Usage

```
TccqLoop(
  id = character(0),
  effect = TccqEffect(),
  body = TccqBlock(),
  semantics = TccqCallSemantics()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.

TccqLoopAxis	<i>Loop-nest iteration axis</i>
--------------	---------------------------------

Description

An axis is one dimension of a loop-nest iteration space: a stable name, a typed extent, and a role. map axes produce output positions; reduce axes fold into an accumulator.

Usage

```
TccqLoopAxis(name = character(0), extent = TccqDim(), role = character(0))
```

Arguments

name	Iteration-axis name.
extent	Typed axis extent.
role	Axis role, map or reduce.

TccqLoopGuard	<i>Selected-arm loop guard</i>
---------------	--------------------------------

Description

A loop guard records one branch decision that must select a loop nest. An ordered list of guards is a control path from outermost to innermost branch; source backends nest those guards around the same TccqLoopNest plan.

Usage

```
TccqLoopGuard(
  condition = TccqExpression(),
  branch = TccqBranch(),
  selected = logical(0)
)
```

Arguments

condition	Scalar logical condition expression.
branch	Source branch payload retaining R special-form semantics.
selected	Whether the consequent (TRUE) or alternative (FALSE) arm selects the guarded loop nest.

TccqLoopNest	<i>Loop-nest program plan</i>
--------------	-------------------------------

Description

TccqLoopNest is the single backend-neutral iteration plan consumed by source printers, in the spirit of the SAC with-loop. One loop nest carries ordered typed axes (map axes produce output positions, reduce axes fold into an accumulator), a value expression or typed statement block whose references carry typed affine accesses, an optional closed reduction plan, and an output access. Elementwise maps, full and per-axis reductions, contractions, stencils, and control-valued results are all instances of this one value; printers must not reintroduce per-family loop shapes or backend-local control trees.

Usage

```
TccqLoopNest(
  id = character(0),
  domain = TccqDomain(),
  axes = list(),
  body = TccqExpression(),
  output = NULL,
  reduction = NULL,
  storage = TccqStorageSlot(),
  guards = list()
)
```

Arguments

id	Stable loop-nest id.
domain	Iteration domain naming the axes.
axes	Ordered TccqLoopAxis values, outermost first.
body	Body expression or typed statement block with access-carrying references.
output	Output access over map axes, or NULL for scalar results.
reduction	Closed neutral reduction plan for reduce axes, or NULL.
storage	Typed storage slot receiving the nest result.
guards	Ordered TccqLoopGuard control path selecting this nest.

TccqLoopTransfer	<i>Neutral loop transfer statement</i>
------------------	--

Description

A transfer is structured control completion, not a read/write side effect. Its action always targets the nearest enclosing sequential loop.

Usage

```
TccqLoopTransfer(
  id = character(0),
  effect = TccqEffect(),
  action = character(0),
  semantics = TccqCallSemantics()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
action	Loop action, either break or next.
semantics	Evaluator facts for the originating R control special form.

TccqLoweredOperation	<i>Lowered operation payload</i>
----------------------	----------------------------------

Description

TccqLoweredOperation is the typed payload attached to operation values after lowering. It keeps operation family, selected implementation, signature, domain policy, and optional reducer facts together so later passes do not branch on loose value attributes.

Usage

```
TccqLoweredOperation(
  family = character(0),
  resolved_op = TccqResolvedOp(),
  signature = TccqOpSignature(),
  domain_policy = NULL,
  elementwise = NULL,
  reduction = NULL,
  contraction = NULL,
  subscript = NULL,
  attrs = list()
)
```

Arguments

family	Lowered operation family.
resolved_op	Selected operation implementation.
signature	Shared operation signature.
domain_policy	Optional result-domain policy.
elementwise	Optional elementwise metadata.
reduction	Optional reduction metadata.
contraction	Optional contraction metadata.
subscript	Optional subscript metadata.
attrs	Structured metadata.

TccqLoweringPlan	<i>Lowering plan</i>
------------------	----------------------

Description

A lowering plan is the typed result of lowering declared R language objects into backend-neutral values, regions, and storage facts. Backend printers consume this plan through TccqProgram; they do not re-discover compiler semantics from source text.

Usage

```
TccqLoweringPlan(
  values = list(),
  result = NULL,
  schedule = NULL,
  regions = list(),
  storage_plan = NULL,
  diagnostics = list(),
  attrs = list()
)
```

Arguments

values	List of lowered IR values.
result	Result value id, or NULL when no expression was lowered.
schedule	Optional ordered program evaluation schedule.
regions	List of executable code regions.
storage_plan	Optional middle-end storage plan.
diagnostics	List of diagnostics attached to lowering.
attrs	Structured lowering metadata.

TccqOpBody	<i>Backend-neutral operation body</i>
------------	---------------------------------------

Description

An operation body is one R expression whose symbols are bound by exact call tags and then positionally through @parameters. Its call index preserves the evaluator facts observed when the body was declared. Lowering expands the body into ordinary typed values before fusion, loop-nest planning, or backend rendering.

Usage

```
TccqOpBody(
  parameters = character(0),
  expression = quote(x),
  call_index = TccqCallIndex()
)
```

Arguments

parameters	Ordered operation parameter names.
expression	Unevaluated R expression implementing the operation.
call_index	Typed calls and evaluator facts for expression.

TccqOpContext	<i>Operation support query context</i>
---------------	--

Description

Operation support query context

Usage

```
TccqOpContext(
  phase = character(0),
  target = character(0),
  region_kind = character(0),
  memory_space = character(0),
  allow_rapi = logical(0),
  allow_boundary = logical(0)
)
```

Arguments

phase	Compiler phase issuing the query.
target	Requested implementation target, or any.
region_kind	Requested execution region kind, or any.
memory_space	Requested memory space, or any.
allow_rapi	Whether implementations touching the R C API are allowed.
allow_boundary	Whether explicit boundary implementations are allowed.

TccqOpImpl

*Operation implementation descriptor***Description**

Operation implementation descriptor

Usage

```
TccqOpImpl(
  op = character(),
  target = character(),
  region_kind = character(),
  memory_space = character(),
  uses_rapi = logical(),
  boundary = logical(),
  pure = logical(),
  effect = TccqEffect(),
  supports = function() NULL,
  render = NULL,
  body = NULL,
  elementwise = NULL,
  reduction = NULL,
  contraction = NULL,
  iteration = NULL,
  subscript = NULL
)
```

Arguments

op	Operation or function name.
target	Implementation target, such as r_language, pure_c, or fortran.
region_kind	Region kind the implementation can run in, or any.
memory_space	Memory space the implementation expects, or any.
uses_rapi	Whether the implementation touches the R C API.

boundary	Whether the implementation crosses a boundary.
pure	Whether the implementation is semantically pure.
effect	Effect summary for calls handled by this implementation.
supports	Predicate receiving a TccqCall and TccqOpContext.
render	Optional source renderer receiving operand strings and a TccqOpRenderContext.
body	Optional backend-neutral operation body.
elementwise	Optional elementwise metadata.
reduction	Optional reduction metadata.
contraction	Optional contraction metadata.
iteration	Optional iteration metadata.
subscript	Optional subscript metadata.

TccqOpImplementation	<i>Operation implementation trait</i>
----------------------	---------------------------------------

Description

Implementations must explicitly opt into this trait before they can be used by an operation registry.

Usage

TccqOpImplementation

TccqOpRegistry	<i>Operation registry</i>
----------------	---------------------------

Description

Operation registry

Usage

TccqOpRegistry(implementations = list(), op_index = list())

Arguments

implementations	List of operation implementations.
op_index	Named list from operation name to implementation positions, used to resolve calls without scanning every implementation. Built by <code>tccq_op_registry()</code> ; an empty index falls back to a full scan.

TccqOpRenderContext	<i>Operation source-rendering context</i>
---------------------	---

Description

Operation source-rendering context

Usage

```
TccqOpRenderContext(  
    language = character(0),  
    backend_id = character(0),  
    attrs = list()  
)
```

Arguments

language	Target source language.
backend_id	Backend requesting rendering.
attrs	Structured rendering metadata.

TccqOpSignature	<i>Operation signature metadata</i>
-----------------	-------------------------------------

Description

A signature is the shared operation contract for argument count and result typing. It may also carry a domain policy for result shape. Elementwise, reduction, and future operation families should carry a signature instead of each inventing their own arity, shape, and type checks.

Usage

```
TccqOpSignature(  
    name = character(0),  
    arity = TccqArity(),  
    result_type = function() NULL,  
    domain_policy = NULL,  
    attrs = list()  
)
```

Arguments

name	Human-readable operation signature name.
arity	A TccqArity contract.
result_type	Function from input TccqType list to result TccqType.
domain_policy	Optional result-shape policy.
attrs	Structured signature metadata.

TccqProgram	<i>Program schema</i>
-------------	-----------------------

Description

Program schema

Usage

```
TccqProgram(  
  values = list(),  
  result = NULL,  
  name = character(0),  
  formals = list(),  
  schedule = NULL,  
  regions = list(),  
  diagnostics = list(),  
  call_index = NULL,  
  storage_plan = NULL,  
  attrs = list()  
)
```

Arguments

values	List of IR values.
result	Result value id or object.
name	Program name.
formals	Named list of formal bindings.
schedule	Optional ordered program evaluation schedule.
regions	List of executable code regions.
diagnostics	List of diagnostics attached to the program.
call_index	Typed frontend call handoff.
storage_plan	Optional middle-end storage plan.
attrs	Structured program attributes.

TccqProgramBody	<i>Backend-neutral program body</i>
-----------------	-------------------------------------

Description

Marker parent for typed structured program bodies. Concrete bodies add lexical scope, statements, and value-completion contracts in the neutral expression layer.

Usage

TccqProgramBody()

TccqProgramOptimization	<i>Program optimization trait</i>
-------------------------	-----------------------------------

Description

Optimizations explicitly opt into transformation and verification. The compiler invokes both through this contract before backend planning.

Usage

TccqProgramOptimization

TccqProgramSchedule	<i>Ordered program evaluation schedule</i>
---------------------	--

Description

The schedule is the semantic owner of top-level R evaluation order. Its linear steps are contiguous after declarations are removed. A structured body is the mutually exclusive control-flow form. In either form, the schedule produces the declared program result. Lowering and backend planning consume this value rather than reconstructing statement order from value ids or source text.

Usage

TccqProgramSchedule(steps = list(), result = character(0), body = NULL)

Arguments

- | | |
|--------|---|
| steps | Ordered TccqEvaluationStep values. |
| result | Result value id produced by the final step. |
| body | Optional structured value-producing body. |

TccqReductionPlan	<i>Closed neutral reduction plan</i>
-------------------	--------------------------------------

Description

A reduction plan owns the reducer, its typed state, the optional condition guarding each transition, one typed assignment per state component, the completed-state projection, and an optional validity expression. No target source appears in this contract.

Usage

```
TccqReductionPlan(
    spec = TccqReductionSpec(),
    state = TccqReductionState(),
    condition = NULL,
    updates = list(),
    value = TccqExpression(),
    valid = NULL
)
```

Arguments

spec	Reduction implementation metadata.
state	Typed neutral state.
condition	Optional scalar logical transition condition.
updates	Ordered state-component assignments.
value	Scalar result projected from completed state.
valid	Optional scalar logical expression proving that value exists.

TccqReductionSpec	<i>Reduction implementation metadata</i>
-------------------	--

Description

A reduction spec is attached to an operation implementation when calls to that implementation can lower to a typed reduction state. Concrete subclasses define that state and its transition; the lowerer and source printers consume the shared protocol instead of recognizing reducer names.

Usage

```
TccqReductionSpec(  
  name = character(0),  
  signature = TccqOpSignature(),  
  associative = logical(0),  
  commutative = logical(0),  
  empty_policy = character(0),  
  attrs = list()  
)
```

Arguments

name	Human-readable reduction name.
signature	Shared operation signature.
associative	Whether the reducer is associative under its declared semantics.
commutative	Whether the reducer is commutative under its declared semantics.
empty_policy	Behavior when no value contributes to the reduction.
attrs	Structured reduction metadata.

TccqReductionState	<i>Typed neutral reduction state</i>
--------------------	--------------------------------------

Description

Typed neutral reduction state

Usage

```
TccqReductionState(components = list())
```

Arguments

components	Named reduction-state components.
------------	-----------------------------------

TccqReductionStateComponent	<i>Reduction-state component</i>
-----------------------------	----------------------------------

Description

A component is one typed scalar in a reducer’s neutral state. Its semantic name is a portable identifier so backend interfaces can derive generated names without sanitizing arbitrary source text.

Usage

```
TccqReductionStateComponent(  
  name = character(0),  
  target = TccqWriteTarget(),  
  identity = TccqLiteral()  
)
```

Arguments

name	Semantic component name.
target	Neutral local target that stores the component.
identity	Initial scalar literal.

TccqRegion	<i>Executable code region</i>
------------	-------------------------------

Description

Regions classify a group of IR values by where they may run and whether they may touch the R C API. kernel, parallel, and device regions are intended to be R-API-free.

Usage

```
TccqRegion(  
  id = character(0),  
  kind = character(0),  
  values = list(),  
  fusion_groups = list(),  
  effect = TccqEffect(),  
  memory_space = character(0),  
  touches_rapi = logical(0),  
  attrs = list()  
)
```

Arguments

id	Stable region id.
kind	Region kind.
values	List of IR values in the region.
fusion_groups	List of fusion groups in the region.
effect	Region effect summary.
memory_space	Dominant memory space for the region.
touches_rapi	Whether the region may call into the R C API.
attrs	Structured region attributes.

TccqRepeat	<i>Neutral repeat statement</i>
------------	---------------------------------

Description

A repeat loop enters its body unconditionally and relies on a typed loop transfer to terminate or begin another iteration.

Usage

```
TccqRepeat(
  id = character(0),
  effect = TccqEffect(),
  body = TccqBlock(),
  semantics = TccqCallSemantics()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.

TccqResolvedOp	<i>Resolved operation implementation</i>
----------------	--

Description

A resolved operation records the implementation selected for one observed call in one operation context. Lowering and middle-end passes should carry this object rather than re-checking support from raw operation names.

Usage

```
TccqResolvedOp(
  call = TccqCall(),
  implementation = TccqOpImpl(),
  target = character(0),
  region_kind = character(0),
  memory_space = character(0),
  uses_rapi = logical(0),
  boundary = logical(0),
  pure = logical(0),
  effect = TccqEffect(),
  body = NULL,
  elementwise = NULL,
  reduction = NULL,
  contraction = NULL,
  iteration = NULL,
  subscript = NULL,
  attrs = list()
)
```

Arguments

call	Observed R call.
implementation	Selected operation implementation.
target	Selected implementation target.
region_kind	Region kind supplied by the implementation.
memory_space	Memory space supplied by the implementation.
uses_rapi	Whether the selected implementation touches the R C API.
boundary	Whether the selected implementation crosses a boundary.
pure	Whether the selected implementation is semantically pure.
effect	Effect summary supplied by the implementation.
body	Optional backend-neutral body supplied by the implementation.
elementwise	Optional elementwise metadata supplied by the implementation.
reduction	Optional reduction metadata supplied by the implementation.

contraction	Optional contraction metadata supplied by the implementation.
iteration	Optional iteration metadata supplied by the implementation.
subscript	Optional subscript metadata supplied by the implementation.
attrs	Structured resolution metadata.

TccqResult	<i>Compiler result value</i>
------------	------------------------------

Description

Compiler result value

Usage

TccqResult(success = logical(0), value = NULL, diagnostics = list())

Arguments

success	Whether the phase completed without diagnostics.
value	Result value for the phase.
diagnostics	List of TccqDiagnostic values.

TccqShape	<i>Shape value</i>
-----------	--------------------

Description

Shape value

Usage

TccqShape(rank = integer(0), dims = list())

Arguments

rank	Number of dimensions.
dims	List of TccqDim values.

TccqStatement	<i>Neutral statement</i>
---------------	--------------------------

Description

Base class for typed statements consumed by source backends. Concrete statement classes own their operands and control structure.

Usage

```
TccqStatement(id = character(0), effect = TccqEffect())
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.

TccqStorageAllocation	<i>Physical storage allocation</i>
-----------------------	------------------------------------

Description

A storage allocation is the physical storage identity assigned to one or more non-overlapping materialized temporary slots. Sharing an allocation is the representation of storage reuse; it is not inferred by source printers.

Usage

```
TccqStorageAllocation(
  id = character(0),
  type = TccqType(),
  memory_space = character(0)
)
```

Arguments

id	Stable allocation id.
type	Semantic storage type shared by every assigned slot.
memory_space	Memory space in which the allocation exists.

TccqStorageLifetime	<i>Storage lifetime interval</i>
---------------------	----------------------------------

Description

A storage lifetime records the definition and last-use positions used by storage reuse planning. It is a typed optimization fact, not source-printer metadata.

Usage

```
TccqStorageLifetime(  
  value_id = character(0),  
  defined_at = integer(0),  
  last_used_at = integer(0)  
)
```

Arguments

- value_id Value whose lifetime is described.
- defined_at Linear definition position in schedule-derived value order.
- last_used_at Last position where the value must remain live.

TccqStoragePlan	<i>Storage plan</i>
-----------------	---------------------

Description

A storage plan belongs to the middle end. It is consumed by backends but does not depend on C, Fortran, TinyCC, or any concrete emitter.

Usage

```
TccqStoragePlan(slots = list())
```

Arguments

- slots List of storage slots.

TccqStorageSlot	<i>Storage slot</i>
-----------------	---------------------

Description

A storage slot records whether a value is materialized, which typed lifetime it occupies, and which physical allocation owns a materialized temporary.

Usage

```
TccqStorageSlot(  
  id = character(0),  
  value_id = character(0),  
  type = TccqType(),  
  role = character(0),  
  materialized = logical(0),  
  lifetime = NULL,  
  allocation = NULL  
)
```

Arguments

id	Stable storage-slot id.
value_id	Value owned by this slot.
type	Slot value type.
role	Storage role.
materialized	Whether storage exists at runtime.
lifetime	Optional typed liveness interval for storage reuse.
allocation	Physical allocation for a materialized temporary.

TccqSubscriptSpec	<i>Subscript implementation metadata</i>
-------------------	--

Description

A subscript spec describes one typed R indexing contract. The first slice is deliberately strict: one scalar integer selector per atomic source axis, with every enclosing iteration proving that its selector is one-based and in bounds. Other selector kinds require their own shape and bounds semantics.

Usage

```
TccqSubscriptSpec(name = character(0), signature = TccqOpSignature())
```

Arguments

name	Human-readable subscript operation name.
signature	Shared operation signature.

TccqSwitch	<i>Neutral positional switch statement</i>
------------	--

Description

A positional switch evaluates one scalar integer selector exactly once, then evaluates at most one ordered alternative. An unmatched position evaluates no alternative, matching `numeric switch()` returning NULL when its value is used only for control. Character selection and value-producing switches require separate representation work.

Usage

```
TccqSwitch(  
  id = character(0),  
  effect = TccqEffect(),  
  selector = TccqExpression(),  
  selector_target = TccqWriteTarget(),  
  alternatives = list(),  
  semantics = TccqCallSemantics()  
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
selector	Scalar integer selector expression.
selector_target	Typed local target for the evaluate-once selector.
alternatives	Ordered positional alternative blocks.
semantics	Evaluator facts for the originating switch special form.

TccqType	<i>Declared compiler type</i>
----------	-------------------------------

Description

Declared compiler type

Usage

TccqType(base = character(0), shape = TccqShape())

Arguments

base	Scalar or storage base type.
shape	Shape value.

TccqValue	<i>IR value</i>
-----------	-----------------

Description

IR value

Usage

```
TccqValue(  
  id = character(0),  
  op = character(0),  
  inputs = list(),  
  type = TccqType(),  
  effect = TccqEffect(),  
  attrs = list()  
)
```

Arguments

id	Stable value id.
op	Operation name.
inputs	Input value ids or references.
type	Result type.
effect	Effect summary.
attrs	Structured operation attributes.

TccqValueBlock	<i>Value-producing neutral statement block</i>
----------------	--

Description

A value block strengthens [TccqBlock](#) by requiring its terminal statement to write one typed result target on every path. Array loop nests and reducers consume this subtype; future procedural control can use the base block without fabricating a result value.

Usage

```
TccqValueBlock(  
    id = character(0),  
    locals = list(),  
    statements = list(),  
    effect = TccqEffect(),  
    result = TccqWriteTarget()  
)
```

Arguments

id	Stable block id.
locals	Ordered local write targets declared by the block.
statements	Ordered TccqStatement values.
effect	Exact effect summary of evaluating the statements.
result	Typed target produced on every terminal path through the block.

TccqValueGraph	<i>Typed lowered value graph</i>
----------------	----------------------------------

Description

This parent owns the value graph shared by lowering plans and programs. It gives expression construction one typed input contract without coupling the neutral expression layer to either pipeline container.

Usage

```
TccqValueGraph(values = list(), result = NULL)
```

Arguments

values	List of lowered IR values.
result	Result value id, or NULL when no expression was lowered.

TccqWhile	<i>Neutral while statement</i>
-----------	--------------------------------

Description

This is sequential recurrence, not a [TccqLoopNest](#). The condition is re-evaluated before every iteration and the body may update explicitly declared mutable cells.

Usage

```
TccqWhile(
  id = character(0),
  effect = TccqEffect(),
  body = TccqBlock(),
  semantics = TccqCallSemantics(),
  condition = TccqExpression()
)
```

Arguments

id	Stable statement id.
effect	Effect summary of evaluating the statement.
body	Procedural loop body.
semantics	Evaluator facts for the originating R loop special form.
condition	Scalar logical condition expression.

TccqWriteTarget	<i>Neutral write target</i>
-----------------	-----------------------------

Description

A write target names a typed value destination without spelling it in C, Fortran, or another target language. Result targets are resolved through a loop nest's output plan; local targets are mapped to generated names by [TccqBackendFunctionInterface](#). The semantic type retains the value domain, while the scalar storage type describes the element written at this target.

Usage

```
TccqWriteTarget(
  value_id = character(0),
  type = TccqType(),
  storage_type = TccqType(),
  kind = character(0),
  binding = NULL
)
```

Arguments

<code>value_id</code>	Stable value id written by the target.
<code>type</code>	Semantic value type.
<code>storage_type</code>	Scalar storage type written at the target.
<code>kind</code>	Target kind: local, result, or cell.
<code>binding</code>	Optional binding owned by the target.

Index

tccq_access, [6](#)
tccq_analyze, [7](#)
tccq_arg_reduction_spec, [7](#)
tccq_arity, [8](#)
tccq_arity_accepts, [8](#)
tccq_assignment, [9](#)
tccq_backend_allocation_binding, [9](#)
tccq_backend_artifact, [10](#)
tccq_backend_context, [10](#)
tccq_backend_error_channel, [11](#)
tccq_backend_extent_binding, [11](#)
tccq_backend_function_interface, [12](#)
tccq_backend_plan, [13](#)
tccq_backend_plan_set, [14](#)
tccq_backend_prepare, [14](#)
tccq_backend_products, [15](#)
tccq_backend_spec, [15](#)
tccq_backend_value_binding, [16](#)
tccq_binding, [17](#)
tccq_binding_reference, [17](#)
tccq_branch, [18](#)
tccq_bridge_plan, [18](#)
tccq_c_backend, [19](#)
tccq_call, [20](#)
tccq_call_index, [20](#)
tccq_call_name, [21](#)
tccq_call_semantics, [21](#)
tccq_cell, [22](#)
tccq_cell_reference, [22](#)
tccq_collect_call_index, [23](#)
tccq_collect_call_semantics, [23](#)
tccq_collect_calls, [24](#)
tccq_compile, [24](#)
tccq_completion, [25](#)
tccq_condition_diagnostic, [25](#)
tccq_conditional, [26](#)
tccq_contraction_spec, [26](#)
tccq_core_backends, [27](#)
tccq_default_op_registry, [27](#)
tccq_diagnostic, [27](#)
tccq_dim_affine, [28](#)
tccq_dim_constant, [28](#)
tccq_dim_equal, [29](#)
tccq_dim_symbol, [29](#)
tccq_dim_unknown, [29](#)
tccq_dimension_reference, [30](#)
tccq_domain, [30](#)
tccq_domain_policy, [31](#)
tccq_domain_policy_result_shape, [31](#)
tccq_effect, [32](#)
tccq_effect_union, [32](#)
tccq_elementwise_domain_policy, [33](#)
tccq_elementwise_expression, [33](#)
tccq_elementwise_result_type, [34](#)
tccq_elementwise_spec, [34](#)
tccq_evaluation_step, [35](#)
tccq_expression, [35](#)
tccq_expression_reference, [36](#)
tccq_expression_tree, [37](#)
tccq_for, [37](#)
tccq_fortran_backend, [38](#)
tccq_fusion_contract, [38](#)
tccq_fusion_group, [39](#)
tccq_if, [40](#)
tccq_index_expr, [40](#)
tccq_index_proof, [41](#)
tccq_indexed_value, [41](#)
tccq_iteration_plan, [42](#)
tccq_iteration_spec, [43](#)
tccq_literal, [43](#)
tccq_literal_finite, [44](#)
tccq_literal_inf, [44](#)
tccq_literal_na, [44](#)
tccq_literal_nan, [45](#)
tccq_local_binding, [45](#)
tccq_loop_axis, [46](#)
tccq_loop_guard, [46](#)
tccq_loop_nest, [47](#)

- tccq_loop_transfer, [47](#)
- tccq_lower_function, [48](#)
- tccq_lowered_operation, [48](#)
- tccq_op_body, [49](#)
- tccq_op_context, [50](#)
- tccq_op_impl, [50](#)
- tccq_op_registry, [51](#)
- tccq_op_registry(), [112](#)
- tccq_op_registry_add, [52](#)
- tccq_op_render, [52](#)
- tccq_op_render_context, [53](#)
- tccq_op_signature, [53](#)
- tccq_op_signature_result_type, [54](#)
- tccq_op_supports, [54](#)
- tccq_opaque_op_impl, [54](#)
- tccq_optimize, [55](#)
- tccq_plan_backend, [55](#)
- tccq_plan_backends, [56](#)
- tccq_program, [56](#)
- tccq_program_loop_nest, [57](#)
- tccq_program_loop_nests, [57](#)
- tccq_program_loop_nests(), [57](#)
- tccq_program_schedule, [58](#)
- tccq_reduction_identity, [58](#)
- tccq_reduction_plan, [59](#)
- tccq_reduction_spec, [59](#)
- tccq_reduction_state_component, [60](#)
- tccq_region, [60](#)
- tccq_registry_supports, [61](#)
- tccq_repeat, [62](#)
- tccq_resolve_call, [62](#)
- tccq_resolved_op, [63](#)
- tccq_result, [63](#)
- tccq_rtinycc_backend, [63](#)
- tccq_shape, [64](#)
- tccq_storage_allocation, [64](#)
- tccq_storage_lifetime, [65](#)
- tccq_storage_plan, [65](#)
- tccq_storage_slot, [66](#)
- tccq_subscript_spec, [66](#)
- tccq_switch, [67](#)
- tccq_type, [67](#)
- tccq_unimplemented_calls, [68](#)
- tccq_value, [68](#)
- tccq_value_block, [69](#)
- tccq_verify_optimization, [69](#)
- tccq_while, [70](#)
- tccq_write_target, [70](#)
- TccqAccess, [71](#)
- TccqArgReductionSpec, [71](#)
- TccqArity, [8](#), [53](#), [72](#), [114](#)
- TccqAssignment, [73](#)
- TccqBackend, [73](#)
- TccqBackendAllocationBinding, [74](#)
- TccqBackendArtifact, [74](#)
- TccqBackendContext, [75](#)
- TccqBackendErrorChannel, [76](#)
- TccqBackendExtentBinding, [76](#)
- TccqBackendFunctionInterface, [77](#), [128](#)
- TccqBackendPlan, [78](#)
- TccqBackendPlanSet, [79](#)
- TccqBackendProducts, [79](#)
- TccqBackendSpec, [80](#)
- TccqBackendValueBinding, [81](#)
- TccqBinding, [82](#)
- TccqBindingReference, [82](#)
- TccqBlock, [83](#), [127](#)
- TccqBranch, [83](#), [88](#)
- TccqBridgePlan, [84](#)
- TccqCall, [85](#)
- TccqCallIndex, [86](#)
- TccqCallSemantics, [86](#)
- TccqCell, [87](#)
- TccqCellReference, [88](#)
- TccqConditional, [88](#)
- TccqContractionSpec, [89](#)
- TccqControlCompletion, [90](#)
- TccqDeadStoreOptimization, [90](#)
- TccqDiagnostic, [91](#)
- TccqDim, [91](#)
- TccqDimensionReference, [92](#)
- TccqDomain, [92](#)
- TccqDomainPolicy, [93](#)
- TccqEffect, [90](#), [93](#)
- TccqElementwiseSpec, [94](#)
- TccqEvaluationStep, [94](#)
- TccqExpression, [95](#)
- TccqExpressionReference, [96](#)
- TccqFoldReductionSpec, [97](#)
- TccqFor, [97](#)
- TccqFusionContract, [98](#)
- TccqFusionGroup, [99](#)
- TccqIf, [100](#)
- TccqIndexedValue, [101](#)
- TccqIndexExpr, [102](#)
- TccqIndexProof, [101](#), [102](#)

TccqIterationPlan, [103](#)
TccqIterationSpec, [104](#)
TccqLiteral, [104](#)
TccqLocalBinding, [87](#), [105](#)
TccqLoop, [105](#)
TccqLoopAxis, [106](#)
TccqLoopGuard, [106](#)
TccqLoopNest, [107](#), [128](#)
TccqLoopTransfer, [108](#)
TccqLoweredOperation, [108](#)
TccqLoweringPlan, [109](#)
TccqOpBody, [110](#)
TccqOpContext, [110](#)
TccqOpImpl, [111](#)
TccqOpImplementation, [112](#)
TccqOpRegistry, [112](#)
TccqOpRenderContext, [113](#)
TccqOpSignature, [113](#)
TccqProgram, [55](#), [114](#)
TccqProgramBody, [115](#)
TccqProgramOptimization, [115](#)
TccqProgramSchedule, [115](#)
TccqReductionPlan, [116](#)
TccqReductionSpec, [116](#)
TccqReductionState, [117](#)
TccqReductionStateComponent, [118](#)
TccqRegion, [118](#)
TccqRepeat, [119](#)
TccqResolvedOp, [120](#)
TccqResult, [55](#), [70](#), [121](#)
TccqShape, [121](#)
TccqStatement, [122](#)
TccqStorageAllocation, [122](#)
TccqStorageLifetime, [123](#)
TccqStoragePlan, [123](#)
TccqStorageSlot, [124](#)
TccqSubscriptSpec, [124](#)
TccqSwitch, [125](#)
TccqType, [126](#)
TccqValue, [126](#)
TccqValueBlock, [83](#), [127](#)
TccqValueGraph, [127](#)
TccqWhile, [128](#)
TccqWriteTarget, [128](#)