SHTAR — שטר

Technical specification for implementers and AI clients

SHTAR — שטר: Symbolic Hierarchical Terms And Rules.

Working draft **0.1**, 25 October 2026.

SHTAR is a financial product definition language. A product is a chronological system of records and payments: observations and decisions establish rights and obligations, and payments occur at their own dates. The definition preserves these relationships over time and specifies its state representation and update rules. Event processing and valuation are interpretations of that definition.

This reference specifies the proposed definition semantics, operational and valuation interpretations, portable representation, compiler responsibilities and conformance rules. For a progression through practical examples, start with the main guide (PDF).

The syntax proposal supplies a machine-readable schema and complete examples. The SPX application overview introduces how product definitions connect to market models.

Contents

1 Status, scope and profiles 2

2 Contractual meaning and operational interpretation 3

3 Values, types and identities 6

4 Typed keys and contexts 8

5 Events, time and processing 10

6 Composition, exports and ownership 12

7 Discretionary decisions 13

8 Registration and numerical valuation leaves 14

9 Proposed syntax and portable semantic IR 16

10 Compilation and backend ports 19

11 Valuation, estimators and hedge interpretation 21

12 Conformance, errors and extensions 23

13 Current implementation and next ports 24

A Framework and artifact metadata 25

SHTAR

1 Status, scope and profiles

SHTAR defines financial products and their contractual meaning independently of an event processor, programming language, simulation model or numerical method. A definition states how records, rights, obligations and payments are related over time, under which conditions they arise, and what information they depend on. It also declares the state representation, initialization and update rules that express these relationships. Choosing individual records or an aggregated state is a choice in the SHTAR definition, not a choice delegated to an interpreter or pricer. Separately defined securities or contexts can compose these relationships. Model independence does not leave the declared state semantics unspecified or permit moving every payment to one terminal date.

Frontends may express definitions through structured text, Markdown, a domain notation or AI-assisted authoring. An operational interpretation applies the definition's declared state and update rules along an accepted history through events and transitions. A valuation interpretation uses that same definition and its information dependencies to value claims. Implementation lowering maps the selected definition and supported interpretations to PyTorch functions, another programming language, an interpreter or a valuation system without choosing a different product-state representation.

The words **must**, **must not** and **may** specify the proposed semantics of this draft. They do not assert that the existing prototype implements the complete language. The portable representation and the current application request format are distinct; Section 9 specifies their exchange boundary.

A **security**, **trade** or **product** denotes a contractual definition of terms, contingent rights and obligations, state and update rules, dependencies and composition. An operational instance binds that definition to actual terms, inputs, identity and current state. A **context** is a product that supplies data and makes no payments throughout its reachable lifecycle. A paying product may also expose an informational context view of its state. A **module** is a reusable definition or component; the `module` kind in this draft is shorthand for a reusable no-payment context.

The core has no SPX restriction, privileged currency asset, common product-state layout, Monte Carlo path dimension, mandatory schedule or required implementation language. Currencies, shares, bonds and other deliverable assets use the same economic treatment. A provider may support a narrower profile and must declare its scope.

1.1 Definition, interpretations and implementation

Four layers have distinct responsibilities:

1. **Product definition and contractual meaning:** terms, declared state representation and update rules specify a chronological system of records, rights, obligations and payments, with contingent relationships, typed context dependencies and composition.

2. **Operational interpretation:** the declared rules are applied to accepted history and decisions to determine the applicable rights and obligations, state changes, deliveries and next processing events. Sections 2–7 give precise rules for this interpretation.

3. **Valuation interpretation:** the context establishes direct or synthetic value status, separately from the observations, models and computational methods used to obtain values. Sections 8 and 11 specify this boundary.

4. **Implementation lowering:** the selected definition and supported interpretations are translated into an operator representation, backend code or another supported execution form while preserving the declared state and update semantics. Sections 9–10 define the proposed representation and compiler responsibilities.

Registration binds a definition and its interpretations to provider capabilities. A compiler profile specifies lowering and transport conventions. A valuation profile specifies numerical interpretation, estimation and hedge reporting. Neither processing code nor a selected pricing method replaces the contractual meaning that it must preserve.

A direct payable has an exchange value accepted or assumed to be market-agreed under the context's liquidity and valuation conventions. A synthetic payable lacks that accepted basis and is valued without assuming market agreement. This value status is separate from computational method: direct and synthetic payables can require model functions, PDE, conditional Monte Carlo or other calculations. It is not determined by asset class, construction or computational cost, and can change with the context while the contract remains unchanged. A security definition must not prescribe a hidden sampler or assume every provider implements every method. Generic estimator and hedge semantics belong to the valuation chapter; application documents state their concrete choices.

2 Contractual meaning and operational interpretation

A definition describes the product before any historical or simulated event is processed. Its meaning includes chronological records, payments, contingent claims, information relationships and the declared state representation and updates. Recording an observation, recognizing an obligation, delivering a right and settling a payment are distinct parts of that history. An authoring surface need not expose the operator tree directly, but the selected SHTAR definition must specify its state and update rules before implementation lowering. A frontend translates those declared rules to the proposed operator and handler representation rather than silently choosing a different representation.

The operational semantics below are normative for that interpretation. They specify how accepted events realize the definition without duplicate effects or loss of contractual information. A valuation backend may use a supported analytic or other interpretation without executing every event step, provided it values the selected definition under the registered conventions; this does not authorize replacing its declared state or update rules with an economically equivalent alternative definition.

2.1 Contractual histories and operational instances

Contractual conditions determine rights and obligations for each realized history. The operational interpretation below applies to one such history. Batching, path cohorts, tensor packing, devices and vectorized branch masks are execution lowerings that must preserve that pointwise meaning.

```
Instance = (
  instance_id, definition_reference, terms, state,
  input_bindings, next_event, state_version, accounting_owner
)
```

A definition reference is versioned and resolved to immutable content. Terms are explicit parameters. State is an arbitrary finite value satisfying the current definition's declared state type, initialization and update rules. Bindings connect capabilities and contexts. Identity, ownership and state version are distinct from the reusable definition.

State may be a scalar, record, array, tagged alternative or reference. A replacement can have unrelated fields, type, dimensions and definition. No common padded state structure is required across different products or realizations.

2.2 Operational processing result

An operational processor interprets the current product instance against an accepted event and information frame:

```
process(instance, event, frame)
  -> transition(deliveries, successor, next_event)
   | pending_decision(decision)
   | error(problem)
```

A successful transition has either an explicit successor and next-event descriptor, or neither. A terminal parent has no successor and no next event. Termination does not cancel previously delivered obligations.

Every continuing review returns its next observation request with its successor. The request identifies a future timestamp or a declared way to select the next eligible event; it is not inferred from a universal schedule. A pending decision remains governed by its separate suspension and resolution rules.

The normal successor preserves the instance ID and advances its state version, even when its definition changes. A newly created payable receives a distinct identity. Transferring an existing right preserves that right's identity and changes its owner. These operations must not be conflated.

2.3 Pure description and atomic commitment

Expressions compute values and declarative outcomes. Constructing a product value does not allocate a live instance, advance a context, post an obligation or sample a market. Unselected branches have no effects.

The coordinator validates an outcome and atomically commits its deliveries, successor and staged context updates. An error commits none of that transaction's staged effects. For a pending decision, it atomically records the request and commits any already validated no-payment dependency updates, but posts no alternative's deliveries or successor. The parent's event remains suspended behind that request; decision resolution uses its captured frame and does not advance those dependencies again. An unresolved dependency cannot supply an after-view or be treated as a completed update.

A transaction may read a previously committed shared update but may not undo it on failure. Updates newly staged by the transaction are committed together or discarded together. Repeated evaluation of an accepted event must be idempotent.

2.4 Chronological records and residual claims

An accepted history retains the identity and timing of relevant observations, decisions, obligations, transfers, settlements and explicit corrections. Calling this history a ledger specifies its accounting meaning; it introduces no new AST field, mandatory storage layout or implemented settlement service. Recording that a dated claim was delivered does not by itself record settlement of the assets due under that claim.

At an information frame t, distinguish settled payments, recognized but unsettled claims, and the continuing rights and obligations contingent on later information. A settled payment remains part of history but is no longer payable under that same obligation. It must not be delivered again, included as an outstanding claim or silently added to a future payment. A partially settled claim retains only its unpaid portion as an outstanding obligation. Its historical record may still affect a later contractual condition. Assets retained from a settled receipt can be separately owned portfolio holdings; their existence does not revive the settled obligation.

A due date earlier than t is not evidence of settlement. A past-due unpaid claim remains a residual claim, subject to its actual settlement, obligor and contractual terms. Missing settlement status is an unresolved requirement; a date cutoff must not silently discard the claim or assume its value. Previously delivered child claims remain outstanding in their owners' economic scope until settled or otherwise explicitly resolved, even if the parent has terminated.

An unpaid coupon-memory entry is not automatically a recognized past-due claim. The terms may

make its eventual payment conditional on a later barrier observation or another event. Such memory represents a contingent entitlement in the continuing product until the contractual conditions create a payable.

A single terminal amount is not generally an equivalent representation of the chronological payments. Carrying an earlier settled receipt to a later date requires a specified holding, reinvestment or financing strategy. The language assumes none. Contractual accumulation rules, when present, belong to the definition; an investor's separate investment strategy belongs to its own declared interpretation or product. Numerical valuation in common units does not create such a strategy or alter the payment dates.

2.5 Declared state choices and averaging

The SHTAR product definition specifies what state it retains and how that state changes. The following coupon definition chooses an aggregate balance explicitly. Interpreters apply this declared choice; it is not a backend optimization. Audit records and payment claims remain separate from the continuing coupon state.

For the memory-coupon component, the chosen state is only the aggregate missed-coupon amount M, initially $M_0 = 0$. At a review before maturity, let c be the current coupon and let I equal 1 when the coupon criterion is met and 0 otherwise. The definition produces a coupon payment amount P and next memory M':

$$P = I(M + c), \qquad M' = (1 - I)(M + c).$$

A nonzero P produces a dated claim under the contractual asset, party and settlement terms; zero produces no coupon claim. That claim is an outcome, not an amount retained in M'. The memory is conditional missed-coupon entitlement, not an already due payment. With $c = 2$, two misses followed by a successful review produce the successive output/state pairs (P, M'): $(0, 2)$, $(0, 4)$, then $(6, 0)$.

Principal redemption is determined separately. If the note continues, its coupon memory is M'; on an automatic call there is no successor, while the resulting claims remain outstanding until settlement. Maturity coupon and redemption terms apply separately. The terms, review events and dates are specified elsewhere; M is the minimal financial memory for this coupon rule, not the entire engine state.

Another authored definition may retain individual coupons. Economic equivalence requires matching eligibility, settlement, accrual and expiry/survival rules, with the same residual rights, dated deliveries and required exports for every continuation; heterogeneous coupons need the distinctions retained. Such state optimization is optional at definition level, and economic equivalence never permits an interpreter or lowerer to substitute a different declared state.

For an equally weighted arithmetic average over contractual fixings $S(t_1), \ldots, S(t_n)$, a SHTAR definition can explicitly declare a running sum U_n, count n and their updates:

$$U_0 = 0, \qquad U_{n+1} = U_n + S(t_{n+1}), \qquad \overline{S}_n = \frac{U_n}{n} \quad (n > 0).$$

The definition's update also increments the count once. Its terms or other declared state retain any required fixing schedule, remaining rights and rounding conventions. This authored sum/count definition exactly represents the contractual finite-fixing average, subject to the declared numerical representation, without rereading all earlier fixings for each new observation. A different SHTAR definition may retain the fixing list and declare averaging over that list; the interpreter must apply whichever definition was selected. A repeated accepted fixing reuses its recorded update rather than incrementing the sum or count again. A correction follows the explicit revision/reconciliation rules.

SHTAR

For valuation after some fixings under the sum/count definition, the accepted historical sum and count are fixed initial state. Each simulated continuation starts from that same declared historical state and applies the declared updates only to its own future fixings. A list-based definition instead starts from its accepted fixing list and applies its own list updates. Valuation paths do not resimulate an already accepted past, reapply historical updates or silently change the state representation. The model's factors and any additional state required by other contractual conditions remain separately available.

A continuously averaged contract instead defines an integral, for example

$$\overline{S} = \frac{1}{b-a}\int_a^b S(u)\,du, \qquad b > a.$$

A SHTAR definition specifies the retained accumulator or other state and its update interface for this continuous-average product. A selected numerical profile may supply a finite quadrature approximation to the integral through that declared interface; it does not discover the product's state or replace its update rules. The quadrature is a numerical approximation, not the exact finite-fixing contract above. Its grid, weights and interpolation belong to the selected numerical profile unless the contract itself prescribes them. The quadrature grid and the model's integration grid have different purposes even when their points coincide; neither may silently replace contractual fixing dates. A seasoned continuous-average contract requires its accepted historical state, including the accumulated integral when declared, or a declared determination of that quantity; missing historical information cannot be filled by simulating a new past. An unspecified product-state choice remains to be resolved in the SHTAR definition, not inferred as a lowering optimization. These distinctions specify representation and interpretation, not a new quadrature or change to the existing simulator.

SHTAR

3 Values, types and identities

3.1 Value domain

The value domain contains null, booleans, integers, finite numbers, strings, dates, instants, durations, party/currency/unit identifiers, finite records and arrays, typed quantities, keys, context snapshots, binding handles, product descriptors, live references, events, wakeups and outcomes. Semantic observation time is a timestamp, represented by the existing `instant` type; `date` remains a distinct calendar value, not an implicit timestamp. The existing `currency` type is an identifier convention in this draft; it does not give currencies a special economic role or a different valuation rule from other assets.

Records and arrays may nest to any finite depth supported by the declared resource profile. Live references allow separate instances to share state without copying it. Tensor and external-buffer transports are optional backend extensions and do not change the value's meaning.

```
any | null | boolean | integer | number | string
date | instant | duration | currency | party | unit
product | reference | binding | wakeup | event | outcome
context | key
{record: {field: Type, ...}, open: boolean}
{array: Type}
{optional: Type}
{union: [Type, ...]}
{quantity: UnitIdentifier}
```

Listed record fields are required. `optional` permits null, not an absent field silently invented by the runtime. An open record permits additional fields; omitted `open` means closed. Array lengths need not be fixed. `any` relaxes a local state interface, not delivery, context, scheduling or registration invariants.

3.2 Numbers and units

A decimal literal is an exact finite decimal encoded as a string and produces type `number`. Integers are a subtype of numbers: integer addition, subtraction and multiplication preserve integer type, mixed numerical arithmetic promotes to number, and division produces number. Booleans and strings are not implicitly promoted. A backend declares its numerical representation and rounding policy. Comparisons use the stated relation; no tolerance, adjusted barrier or replacement of a printed level by a nominal percentage is implicit.

Arithmetic is defined over finite mathematical values. Division by zero, operations outside their real-valued domain, overflow in the selected execution profile and nonfinite final values produce explicit errors. Contractual rounding must be an explicit bound operation.

Quantities carry units. Addition and ordering require compatible units; multiplication and division follow the registered unit algebra. A number alone does not establish whether it is a fixing, an amount due later or a present value.

Contractual denomination and delivery assets are part of the product's terms. The reporting asset or pricing numeraire is selected by a valuation interpretation. Changing that selection does not redenominate the contract, replace what must be delivered or change an obligation's quantity or date. Any contractual conversion or asset-selection rule must be explicit in the definition.

For assets or claims A and B, write $X_t(A, B)$ for the exchange value of one unit of A in units of B at the accepted valuation frame t. Thus a quantity q of A has exchange value $qX_t(A, B)$ in B units when that exchange value and its conventions are supported. The frame includes the information boundary; the asset or claim identity includes the relevant rights, settlement and obligor. A currency, share or bond may serve as either argument. Direct status means the context accepts or assumes market agreement on the exchange value under its stated conventions. Obtaining or projecting that value may still require a model. Synthetic status means this accepted market-agreement basis is absent; a model valuation alone establishes neither agreement nor an executable quote.

A required direct conversion can use a supplied context-accepted exchange value, a pure derivation from compatible accepted values, or a registered model computation of the accepted valuation relationship. It does not require a separately stored table of every asset pair or a closed-form formula. For example, conversion through a common direct asset N can use

$$X_t(A, B) = \frac{X_t(A, N)}{X_t(B, N)}$$

when the bound exchange conventions support this composition and the denominator is nonzero. Differing settlement rights or quote conventions cannot be silently identified. Missing conversions remain unresolved. A model-computed exchange value retains its provenance when exposed under a typed key; computation or exposure alone neither establishes nor precludes the market-agreement basis required for direct status in Section 8.

3.3 Definitions, descriptors and references

A definition is reusable immutable content. A product descriptor binds it to terms, state, inputs and an initial wakeup without allocating ownership. A live reference identifies an existing instance and the rights or capabilities granted to its holder.

Two fresh materializations of a descriptor create distinct instances. Two references to an instance share state. Reading its context is not a transfer of its rights. A constructor with a current-state override must also provide its corresponding next-event descriptor; otherwise the definition's initializer supplies both.

A `product` expression evaluates its supplied terms and binding handles into a pure constructor descriptor. If explicit state and next expressions are supplied, their values bypass initialization. Otherwise the initializer is evaluated when the coordinator stages that descriptor as an entry, successor or delivered instance, using the child's terms/bindings and the accepted event/frame. The intended instance identity is determined by the entry, successor or delivery identity rule before initialization; no ownership is committed yet. Initialization has no old state or `self` and must be pure. A logically repeated materialization reuses the same initialized result or reproduces it identically.

In a handler, `self` is the immutable descriptor of the current instance. `replace(product, state)` changes only its state and never reruns initialization. In a transition the explicit `next` field is authoritative for the successor, replacing any wakeup carried by its descriptor; no second wakeup is retained. A newly delivered child keeps its own initialized or explicitly supplied wakeup.

3.4 Resolution and reproducibility

A document specifies its SHTAR version, namespace and imports. Imports pin namespace and version; the resolver records content digests and verifies any supplied digest. Within the applicable registry, an exact local declaration is checked first, including a locally declared dotted name such as `market.close`. Otherwise unqualified names resolve in the current namespace and qualified names through that namespace or explicit imports. A declaration and an import alias that would make resolution ambiguous are rejected. Definitions, key types and pure functions have separate registries; no implicit global fallback is permitted.

JSON object-member order has no semantic effect. Duplicate keys are invalid. Arrays preserve declared order. Exact decimal spelling remains available in source provenance even when two spellings denote the same number.

Reproducible artifact identity covers checked IR, required artifact metadata, resolved imports, compiler profile and generated artifact. A hash identifies bytes or a named canonical encoding; it is not a signature or proof of authorship.

SHTAR

4 Typed keys and contexts

4.1 Typed logical multimap

A key declaration associates a parameter signature with a result type and permitted multiplicity:

```
KeyType(parameters) -> Key[ResultType]
```

A concrete key is its versioned key-type identity plus canonical typed arguments. Time, asset, source, currency, instance or scenario may be arguments. The returned data's type is determined by the key type, not guessed from its name or current contents.

At an information frame and phase, a context is logically:

$$\Gamma_{t,\phi}(k) \ \subseteq_{\text{multi}} \ \text{ValueType}(k).$$

This typed multimap need not be a materialized dictionary. A provider may resolve parameterized keys lazily. Each entry carries producer identity and snapshot provenance.

For example, `market.close(asset)` can return a number or quantity; `context.market()` can return a context; and `product.current_state(instance)` can return a declared variant. Similar strings with different key-type identities are different keys.

4.2 Lookup and multiplicity

`lookup(context, key, one)` requires exactly one entry. Zero means missing data; more than one means ambiguity. Neither may be silently replaced with zero, null, the first match or the last match.

`lookup(context, key, all)` returns all resolved entries and preserves multiplicity. It may return an empty collection for a supported, fully resolved key with no entries. An unavailable provider or unresolved dependency is still an error.

Registry cardinality `one` permits at most one entry per bound concrete key; `many` permits several. It does not require every context to contain every possible argument of a parameterized key. The consuming lookup and registration establish required presence.

Each provider or overlay entry contributes one value of the key's result type. If that type is an array, `all` returns an array of arrays rather than flattening it. Equal values from distinct producer entries remain distinct. A deterministic transport order must not become an implicit priority policy.

4.3 Nested contexts

A key may return a context snapshot. Nested lookup first resolves that context and then looks up a key within it. Nested namespaces are not implicitly flattened.

Nested views remain tied to the allowed frame and phase. Returning a live reference must not let a later lookup expose future state through an earlier snapshot. A provider must pin the view or reject the unsupported snapshot operation.

`context(port)` returns the immutable read view at the consumer's current frame and phase. Completed declared dependencies are visible in that view; an unresolved dependency or future-phase version is not. An explicitly bound historical or before-view remains pinned to that earlier version. With a plain live-instance binding, the view uses the applicable staged update when present, otherwise the latest committed state permitted by the current frame. This rule also applies to typed lookups after a declared synthetic-context update. The `export` operator additionally supports an explicit before/after selector at a named phase.

4.4 Derived contexts

A stateless derived context is a pure function of explicit parent contexts, state and accepted values. A stateful derived context is a no-payment product with ordinary state, event handlers and next-event descriptors. Derived or synthetic state is not itself a payable classification: it may represent wholly observable calculations or model-produced information. Model-produced information retains its model dependency when consumed through a context.

`context_overlay` combines parent snapshots with explicitly computed key/value entries. Its collision policy is mandatory:

- `error`: multiple entries for a resolved concrete key are an ambiguity.
- `collect`: retain all entries, subject to the key's permitted multiplicity.

Repeated paths to the same snapshot entry are aliases, deduplicated by entry identity. Distinct entries with equal values remain distinct. There is no implicit overriding or last-write-wins rule. Selecting or aggregating values requires an explicit derivation.

4.5 Contexts as products; paying-product views

A definition of kind `context` or `module` must emit no payments or transfers in any reachable transition or decision alternative. Its continuing successor must remain a context. Its state shape, inputs and

future events may change.

A paying security may expose a read-only context facet of its state. The economic owner remains responsible for its payments. Consuming that facet does not acquire the security, discard its cashflows, recreate it or process it twice.

The information-dependency graph and the ownership/delivery graph are separate. A context may expose a security descriptor or numerical value as data; doing so creates no owned payable.

5 Events, time and processing

This chapter specifies event and time rules for the operational interpretation of a product definition. Event handling realizes contractual conditions and changes; it does not define the language's purpose or require every authoring surface to expose handlers.

5.1 Accepted event envelope

An accepted event has immutable `id`, `kind`, `when`, `payload`, `revision` and `source` fields. Its `when` is a resolved timestamp of type `instant`. The coordinator supplies the corresponding `context.when`, `context.instance_id`, `context.frame_id`, `context.phase` and `context.owner`. Initializers see terms, bound inputs, event and context; handlers additionally see state and the immutable descriptor `self`.

Economic event time differs from record-receipt time. Historical correction requires an explicit revised history or reconciliation process, never silent mutation of committed history.

Consumers can share an event ID. Equal dates or timestamps alone do not identify one event. Distinct ticks can share the available timestamp resolution; a registered source order and frame identity must establish their strict logical order. Different IDs alone do not establish which event is later. Derived dependency requests inherit the original accepted event identity and economic timestamp; they cannot fabricate historical market observations.

SHTAR

5.2 Returned wakeups

`event.at(when, kind, payload)` constructs a scheduled wakeup at a timestamp strictly later than the current event's timestamp. A date supplied as authoring convenience must first be resolved under the explicit conventions in Section 5.3. Same-time transformations belong inside the transaction or explicitly ordered phases, not an `at` self-rescheduling loop.

`event.await(kind, not_before)` subscribes to an externally accepted event under a timestamp gate. It does not invent an arrival distribution or advance time. The subsequent event must satisfy the gate and strictly advance the registered logical event order without replaying the current event. `not_before` is an inclusive timestamp bound; it does not alone express an instruction to observe every available tick.

`event.epsilon(kind)` is the proposed immediate next-observation request: select the first eligible accepted event of that kind after the current observation in the registered source order. Here **epsilon** means as frequently as eligible observations become available, or the next available tick. It is neither a literal real infinitesimal nor a fixed numerical time increment. There is no minimum elapsed gap. A tied timestamp is permitted only when the registered source order and information frame strictly advance; an identical event or an unordered timestamp tie cannot satisfy the request. Normal observation order must not move backward in economic time; historical corrections use the explicit reconciliation rules.

Constructing this relative wakeup is pure. The coordinator resolves it against the bound event source; the constructor does not sample a path, read a future fixing or manufacture a clock step. If no eligible event has arrived, the request remains awaited. A subscription can implement it only when the registered source guarantees these first-eligible-event and ordering semantics. Neither `after` export views nor adding zero to a timestamp express this wakeup.

A supported valuation profile may interpret an epsilon-driven observation rule through a continuous-observation model or an explicitly declared numerical approximation. Epsilon does not itself change the model's clock, integration grid or step count. The profile must state the observation interpretation and its adequacy for the contract; hourly, tick-based and continuous monitoring are not universally interchangeable, particularly for barriers and other path-dependent conditions.

An initializer may return null for a dormant context supplying static data. It receives no events until explicitly activated or subscribed. An active security must declare scheduled, awaited or epsilon processing. In a committed transition, null successor and null next mean termination; a nonnull successor requires a nonnull wakeup returned by the definition.

Registered external or exceptional events may interrupt a scheduled wakeup when the definition declares a handler and the provider binds the source. The resulting transition replaces the old wakeup. A later scheduled timestamp does not silently suppress an accepted interruption.

5.3 Timestamps, calendar dates and model time

SHTAR observation time is a timestamp denoting an instant. Calendar dates remain useful contractual inputs and authoring conveniences, but a date-valued review or settlement instruction requires a declared fixing or settlement time, calendar and time zone to resolve its timestamp. Calendar identities and versions are bound dependencies. A local time requires an explicit zone and any necessary daylight-saving resolution; there is no implicit midnight, market close or host-machine time zone. Mixed date and timestamp inputs are compared only after the required resolution. Missing or ambiguous conventions remain unresolved.

Mapping a resolved timestamp to a floating-point model time, year fraction, integration index or clock coordinate belongs to the provider/model, not to the product definition. It must preserve the established model's time map, clocks and grid conventions. This does not remove contractual day-count or accrual rules: a coupon's stated day-count convention remains a product term and may differ from the model's numerical time coordinate.

SHTAR

Processing time, delivery of a right and settlement are distinct. A product can deliver now a claim that pays later. A promise to transfer a security later is a future obligation, not an immediate transfer with an ignored date.

Settlement must be established from accepted records or a bound settlement convention. Passing the scheduled settlement date alone does not establish that the obligation was discharged. Chronological replay preserves this distinction when constructing the residual claims at a later frame.

Calendar date rules and timestamp rules are expressions over accepted state and explicit bound conventions. Missing calendars/fixings and unsupported timestamps are unresolved requirements. Contractual observation timestamps must not be silently rounded to a simulation grid or shifted to a convenient business day. A date-only application profile must declare how its supported dates are interpreted; it does not establish general intraday or epsilon support.

5.4 Transaction and replay

The coordinator:

1. Checks the event, information frame, handler, ownership and registered capabilities.
2. Resolves dependency order and stages compatible shared-context updates.
3. Evaluates the selected pure handler body.
4. Validates the complete outcome, successor, wakeup and deliveries.

5. Commits once, records a pending decision with only its validated no-payment dependency updates, or reports an error with all newly staged effects discarded.

Delivery identity combines parent instance, event identity including revision, and a local delivery key. Replay reuses the committed outcome rather than reapplying accumulator updates or reposting deliveries. It must not reset later recorded settlement status or settle the same obligation twice. A conflicting replay with changed terms, amount or dependency payload is an error. Resuming from accepted sufficient state processes only subsequent events; replaying from an earlier state must reconstruct the same history without reposting effects to an existing ledger.

Dependency closure and event ordering must be reproducible. A resource limit may produce an explicit error; it may not silently truncate events or approximate the contract.

6 Composition, exports and ownership

6.1 Separately defined securities as payment

A delivery may construct a security from another versioned definition, with its own terms, bindings and current state. It remains security-valued until the provider chooses a valuation route.

The proposed delivery modes are:

- `create`: materialize a new claim from a descriptor and assign it to a recipient.
- `transfer`: transfer an owned live right, preserving its identity and consuming the transfer capability.
- `asset`: a quantity of an identified asset, with obligor, recipient and settlement.

SHTAR

These are delivery modes, not direct/synthetic value statuses. The first two are not abbreviations for cash settlement. A separately defined swap, option, bond or other security may be the payable; currency quantities have no privileged treatment. Direct or synthetic status depends on the context's accepted market-agreement and liquidity basis at the relevant frame, separately from whether computing its value requires a model.

An asset's observed exchange value for immediate delivery does not by itself value a promise to deliver it later. The actual payable includes its settlement and obligor conventions. Its acceptance basis and computational route must match that claim; a missing spot-to-deferred conversion does not itself determine direct or synthetic status or authorize a replacement model.

A positive asset quantity flows from obligor to recipient. A negative quantity must be normalized by reversing those parties and taking the absolute amount before binding the obligation. Its actual credit and settlement meaning must be supported.

6.2 Shared state and dependency phases

A handler declares dependencies on bound contexts/modules by port, derived event kind, payload and phase. An optional guard is evaluated from available pre-update information.

The coordinator stages a live instance at most once per accepted event. Repeated compatible requests reuse its response. Conflicting payloads, phases or update rights for the same instance/event are errors. Lookup itself never advances state.

Ordered phases and dependency edges determine visibility. An `export` selects `before` or `after` at a named phase. `before` is the applicable pre-update snapshot. `after` requires the matching staged or committed response; missing `after` data never falls back to old state.

Within a phase, after-dependencies form an acyclic graph. A cycle is invalid unless an explicitly prior snapshot breaks it. Independent operations may be reordered only when their observable result is identical.

6.3 Named exports and typed-key providers

A definition can publish named exports and typed-key providers. Named exports are a component convenience; typed keys implement the general context abstraction. Neither automatically denotes a price. Exposing an exchange value preserves its acceptance basis and observable or model provenance; model dependence does not itself determine direct or synthetic status. Neither view creates ownership of its referenced asset or claim.

In a `provides` rule, concrete key arguments are visible under lexical root `key`. The pure value expression produces the key's declared result type. Unsupported arguments can produce an explicit error. Multiple matching rules are not silently prioritized.

Constructors bind inputs explicitly. `binding(port)` forwards a capability without turning it into an arbitrary string or duplicating a live instance. Read and transfer capabilities are distinct.

6.4 Replacement types

A successor can be the current descriptor with changed state or a descriptor for another definition. Only the new definition's interface applies to the new state; old fields need not survive.

If replacement changes exported information's type, it must use a correspondingly typed key or an explicit variant. The same key cannot silently return another type. Incompatible consumers must re-register or fail.

SHTAR

7 Discretionary decisions

`pending_decision` identifies a stable decision ID, deciding party, deadline, named alternatives and an optional explicit default. Alternatives are lazy outcome expressions and can deliver products, replace state or terminate.

Without an accepted action, processing yields a pending request. No alternative is selected, averaged, assigned a probability or booked by accounting. The request retains event/frame identity and the necessary terms, state and lexical environment.

An accepted action must match the ID, authorized party, allowed action and timing. The coordinator resumes or replays against the recorded information frame and commits the selected alternative once. If later information is required, the decision must be represented at the later event rather than reading future data through an earlier frame.

A default exists only when contractual. Deadline passage is an accepted event or determination, not an implicit choice by the language. Revised decisions require explicit reconciliation.

Historical accounting can process a choice without modeling why it was made. Valuation requires a separately registered policy or solver. Discretion is not identified with an exogenous random market shock, and the core does not require an exercise model.

8 Registration and numerical valuation leaves

Registration connects a product definition to supported operational and valuation interpretations. It resolves capabilities and conventions without changing the definition's contractual rights or obligations.

8.1 Capabilities

Registration resolves definitions, key types, contexts, event sources, calendars, accepted historical state, residual claims, projection support and possible delivery kinds. It distinguishes settled history from unsettled claims and binds any required historical accumulation and settlement evidence. It binds the reporting asset separately from contractual denominations and deliveries, records the value-status acceptance basis and its conditions, and resolves observable and model dependencies for the computational route. It selects supported routes and returns `ready`, `unresolved` or `unsupported` with issues and provenance.

A model may recognize a delivered product and register a conditional valuation function for it. Registration checks that the function values that product's actual terms, declared state and obligations at the required information frame; recognition does not replace the product definition.

`unresolved` means required data or bindings are absent. `unsupported` means a capability is not implemented. Neither authorizes a substitute model, product or method.

Constructors, predicates and ranges can cover future dynamic states and dates without enumerating every possible shape. Actual lookups, successors and deliveries must satisfy those registered capabilities. Value status and routing conditions must be re-evaluated at the relevant frame or context change; earlier direct acceptance is not a permanent property of the product identity.

The information boundary is explicit. Market observations must be available at the accepted frame. A conditional value can depend on future payoffs using current information; it is not access to a future realization. An asset directly quoted today need not have a known exchange value at a future frame. Projecting that future observation is a model responsibility even when the realized asset is directly interpretable at that later frame.

8.2 Value status and exclusive computational routes

A payable is **direct** at a particular frame when the bound context accepts or assumes market agreement on its exchange value under stated liquidity and valuation conventions. This can apply to a constructed SHTAR bond or swap, a liquid option, or another supported claim. Agreement need not establish one universal price or a literal executable quote. The context or profile records the acceptance basis, relevant quote conventions such as bid/ask side, and liquidity suitability. The basis must match the actual claim's rights, settlement and obligor. The core imposes no numerical liquidity threshold or automatic liquidity assumption based on asset class.

A payable has **synthetic**, or **indirect**, status when that accepted market-agreement basis is absent. Its value is assessed under the chosen model and assumptions without presumed agreement among market participants. Different models can value the same defined claim differently. Merely storing, caching or naming a private model result cannot establish direct status or erase its provenance. The same bond or option can migrate from direct to indirect when its context no longer supports the required agreement or liquidity basis. An off-the-run bond or previously issued option is not automatically indirect merely because of its age; the acceptance conditions decide. The contractual definition, state and obligations remain unchanged by this status change. Status is contextual, not a permanent definition kind or an additional syntax field.

Value status and computational method are separate. A direct bond or swap in a short-rate model, or a liquid option or VIX future/option accepted as direct by its context, may still require substantial

model computation. Neither an analytic function nor PDE or conditional MC implies synthetic status. Registration may select one computational route for either status:

- Use a supplied exchange value or a compatible observable derivation.
- Evaluate a registered model value function or numerical continuation of the product's definition, terms and state.
- Explicit unresolved/unsupported status.

A model route may use an analytic function, PDE, branching or conditional MC, or another supported method while preserving conditional market context and dependence. An available valuation function can evaluate a recognized child without expanding its dated payments. No closed-form representation is assumed, and lack of one does not prescribe PDE or any other method. Where current accepted values are supplied, the profile specifies the model representation's required matching or calibration conditions and reports discrepancies; direct status alone does not assert numerical agreement or permit silent substitution of an inconsistent model output.

The function's conditioning inputs must be sufficient for the actual claim. Current model state alone is sufficient only when the selected model and product justify it; otherwise required product-state snapshots or accepted history must also be bound. The provider may not invent a Markov property or omit relevant history. Nested valuation may not advance the parent's live state or consume its random stream without a registered equivalent mechanism.

An obligation contributes once, whether through an observed value, a derived value or a model value. Its selected value and its expanded future receipts must not both contribute. Accounting settlement records and valuation representations may coexist, but their identities and price contributions must remain distinct. A missing input, lapsed acceptance basis or unsupported computational capability requires explicit resolution or re-registration. A change to indirect status does not silently authorize another solver or method; absent the required registered capability, the result is unresolved or unsupported.

SHTAR

8.3 Numerical leaves

After route selection and conversion to units of the selected reporting asset B, terminal contributions are numbers:

$$V = \mathcal{A}_\Pi(v_1, \ldots, v_n), \qquad v_i \in \mathbb{R}.$$

The reporting numeraire can be any supported direct asset, including a share, bond or currency. This choice does not change the contractual denomination or deliverable asset. The valuation profile determines aggregation, conditioning, intertemporal conversion and numerical method. A numerical leaf can result from an observed, derived or model-computed contribution under either value status, including one reached by further product processing. Its numerical form does not erase its acceptance basis or computational provenance. Vectorized execution may carry one number per realization.

A future delivery of 100 units of A is not automatically a present-value leaf of 100 units of B, even when $A = B$. Asset and claim identity, settlement, obligor, market context and valuation convention must first be resolved. No currency-specific discounting primitive is implied by the core.

This requirement applies to the economic valuation graph. Contexts and states may contain dates, identifiers, other contexts, products or decisions. A numerical context lookup does not itself create an asset or an additional price contribution.

Numerical leaves are outputs of a valuation interpretation of the selected residual claims. They do not replace the chronological payment records with a new terminal contractual payment, and settled historical payments do not become new leaves in the remaining product value. A separate historical-performance calculation must state its scope and any carrying or reinvestment assumptions.

9 Proposed syntax and portable semantic IR

9.1 Authoring and canonical exchange

The proposed portable representation is a tagged JSON AST. Its complete structural grammar is the JSON Schema. It represents terms, dependencies, composition and explicit operators, including handlers for the operational interpretation. It is an exchange form and compiler boundary; authors need not express the original product definition as raw JSON or manually written event handlers.

A conventional frontend or an AI-assisted compiler can translate selected definitions, including their state representations and updates, into this operator format, with explicit source-to-rule reconciliation and semantic checks. If a source term sheet does not choose a state representation, authoring a SHTAR definition makes that choice explicitly before lowering. This draft does not supply a separate standardized declarative parser or an implemented general compiler. The existing structural schema checks the proposed operator representation; it does not itself translate prose or prove that an operational lowering preserves the source contract and declared state semantics.

The portable document uses the literal version field `sdl` with value `0.1-draft` and schema identifier `urn:sdl:syntax:0.1-draft:public`. The current application request format `sdl-pricing-0.1` is separate: its parser does not accept the proposed portable representation.

An optional compact frontend could expose an operational form such as:

```
security ActivateChild(terms)
  uses market: Context, synthetic: Context
  on review:
    s := one(market, Market.Close(asset = terms.asset))
    a := one(synthetic, Accumulator.Level())
    if s >= terms.trigger:
      pay create ChildSecurity(terms.child, state = {level: a})
          to terms.recipient key "child"
      stop
    else:
      become self with state {last: s, accumulated: a}
          next epsilon("review")
```

This is an illustrative operational surface-syntax proposal, not a standardized parser grammar or the required authoring form for every product definition. A lowering to the typed JSON IR must preserve the source contract and satisfy the operational semantics. Complete descriptors, bindings and wakeups must be resolved even when source notation elides them.

9.2 Document and definition forms

The root contains `sdl`, `namespace`, required `attribution` (Appendix A), versioned `imports`, ordered `phases`, a definition map, and optional key declarations and entry. Descriptive metadata is separate.

```
Definition:
  kind          product | context | module
  parameters    name -> Type
  state_type    Type
  inputs        port -> InputCapability
  exports       name -> {type, value_expression}
  provides      optional typed-key provider rules
  init          {state_expression, next_expression}
  on            event_kind -> {phase, dependencies, body_expression}
```

Input capabilities declare kind and type, with access and availability where relevant. General `context` inputs are preferred. `observation`, `valuation`, `calendar`, `event` and `module` ports are specialized provider conveniences, not a privileged built-in market implementation.

The existing `state_type`, `init` and handler body expressions express the authored state choices and updates described in Section 2.5. Arbitrary records, arrays and expressions already support these alternatives; they require no new schema field, optimizer flag or special aggregate operator.

An entry names a definition, a terms expression and external bindings. Source bindings identify provider-owned capabilities; instance bindings identify registered live instances. Instance/export bindings also specify view and phase.

9.3 Expression forms

Every AST node has an `op` discriminator. Records are explicitly constructed; a business field named `op` does not become executable.

```
literal          {type, value}
get              {root, path}
record           {fields: name -> Expr}
array            {items: [Expr]}
if               {condition, then, else}
let              {bindings: [{name, value}], body}
call             {function, arguments: [Expr]}
fold             {items, initial, item, accumulator, body}
input            {port, arguments?}
binding          {port}
reference        {port}
export           {port, name, view, phase}
key              {name, arguments: name -> Expr}
context          {port}
lookup           {context, key, selection}
context_overlay  {parents: [Expr], entries: [{key, value}], collision}
product          {definition, terms, bindings, state?, next?}
replace          {product, state}
transition       {deliveries, successor, next}
pending_decision {id, party, deadline, choices, default?}
fail             {code, message}
```

The schema fixes fields, literal types and paired `state`/`next` and `successor`/`next` requirements. Unknown semantic fields/operators are invalid. Metadata cannot alter semantics.

`get` roots are `terms`, `state`, `self`, `event`, `context`, `key` where available, or a lexical binding. Paths contain field names or nonnegative indices. Missing fields and invalid indices fail. `let` binds sequentially; redefining reserved roots or existing local names is invalid. `fold` iterates a finite array in order with named item and accumulator.

`if` evaluates only its selected branch and requires `else`. Function arguments are evaluated before a

pure call. Conditional availability must use `if` rather than an undocumented short-circuit function.

9.4 Pure operations

The core registry fixes names, arity and type rules:

- `core.add`, `core.sub`, `core.mul`, `core.div`: binary arithmetic with compatible units; division by zero fails.
- `core.max`, `core.min`: binary ordering of compatible values.
- `core.eq`, `core.ge`, `core.gt`, `core.le`, `core.lt`: typed equality/ordering without implicit string or missing-value coercion.
- `date.add_days(date, integer)` and `date.add_business_days(date, integer, calendar_binding)`.
- `event.at(when, kind, payload)`: scheduled timestamp wakeup, with any date shorthand explicitly resolved before use.
- `event.await(kind, not_before)`: external-event subscription with an inclusive timestamp gate and strict logical-event progress.
- `event.epsilon(kind)`: pure relative wakeup for the next eligible tick under the source ordering in Section 5.2; no numerical epsilon argument.
- `quantity.make(amount, unit)`: construct a typed quantity.
- `quantity.scale(quantity, number)`: multiply by a dimensionless number.
- `quantity.ratio(quantity, quantity)`: form a dimensionless ratio of compatible quantities.

SHTAR

These core namespaces are fixed intrinsic registries. Additional functions come from explicitly resolved versioned registries with defined types, purity and information requirements. Arbitrary host-language source, filesystem/network operations, dynamic imports and unrestricted `eval` are not expressions.

The proposed epsilon constructor uses the existing generic `call` node with `function` set to `event.epsilon` and one string argument for the event kind. No new AST field or structural schema variant is required. Structural acceptance of that call does not assert an implemented interpreter or supported tick source; semantic linking and provider registration must recognize the operation and its ordering contract. Timestamp literals use the existing `instant` type, while `date` literals remain calendar inputs.

9.5 Typed lookup syntax

```
{
  "op": "lookup",
  "context": {"op": "context", "port": "market"},
  "key": {
    "op": "key",
    "name": "market.close",
    "arguments": {
      "asset": {"op": "get", "root": "terms", "path": ["asset"]}
    }
  },
  "selection": "one"
}
```

The key registry fixes argument and result types. The bound context supplies data at the allowed frame. This expression knows no particular data source or stochastic model.

9.6 Complete reference fixtures

Four complete structurally checked documents accompany the proposal:

1. `01_product_delivery.json`: a security creates a separately defined security as payable.
2. `02_shared_running_context.json`: two readers share one staged synthetic-context update and return state-dependent dates.
3. `03_pending_decision.json`: a party's alternatives remain uncommitted pending an accepted decision.
4. `04_nested_derived_context.json`: context-valued keys, nesting, a derived overlay and preserved multimap multiplicity reach a numerical payoff.

These are reference IR fixtures. Metadata describing external instance registries is setup documentation, not allocation instructions. The existing application request parser does not accept them.

10 Compilation and backend ports

10.1 Pipeline

```
Source terms + explicit state/rule choices
              |
       Frontend, possibly AI
              |
   Selected SHTAR definition in typed IR
              |
   Link + type/effect/context checks
              |
   Provider registration and lowering
              |
  PyTorch / other language / interpreter
```

An AI may be a frontend, translator or compiler from a financial product definition to the proposed operator and handler format or a checked backend lowering. Its output must be reviewable and satisfy the same obligations as a conventional compiler. Generated event handlers and pricing code are implementations of an interpretation; they do not replace defining the contract's meaning. This pipeline is a specification of responsibilities, not a claim that the general compiler already exists.

An AI or author may propose an alternative SHTAR definition with aggregated coupons or running-sum averaging and explain its economic equivalence to a detailed definition. Selecting that alternative is an explicit definition-authoring step before lowering. It is not an implicit compiler rewrite, and a provider must not make that substitution merely because it is faster or uses less memory.

The compiler records source-to-IR mappings, unresolved clauses, accepted assumptions, imports and backend profile. Ambiguous rights, calendars, inequalities and unidentified state remain explicit rather than being filled with plausible defaults.

10.2 Backend responsibilities

```
resolve definitions, keys and pure operations
construct and validate product/context descriptors
register observations, events and delivery capabilities
read typed keys at an allowed information frame
stage and memoize context dependencies
process events into outcomes
commit accounting effects exactly once
value each delivered claim through one selected route
reduce valuation contributions to numerical leaves
report provenance and diagnostics
```

These are responsibilities of the supported operational and valuation interpretations, not required Python method names or the defining form of every source contract. PyTorch may pack declared records into tensor storage and vectorize the declared rules across realizations. Other backends may use objects, generated functions or PDE representations. Such physical storage and execution choices preserve the selected definition's state values and updates; they cannot replace a declared list by a newly invented accumulator. All preserve contract and information semantics.

10.3 Preservation

A conforming lowering preserves the selected definition's declared state representation, initialization and updates, as well as comparisons, rounding, event and settlement conventions; identities, ownership and obligation keys; the distinction between settled history and residual claims; typed-key multiplicity and snapshot visibility; lazy branches and pending alternatives; the distinction between reads and owned claims; and exclusive computational routes, value-status acceptance conditions, observable and model dependencies, and numerical units. An aggregated state is used only when the selected SHTAR definition declares it. A lowerer must not introduce a different state representation on grounds of economic equivalence, whether or not that state is exported. Audit history remains preserved separately. Exact contractual fixing accumulation remains distinct from numerical approximation of a continuous averaging rule.

Storage packing, scheduling and common-subexpression optimizations are allowed only when they preserve the declared state values and update semantics as well as observable results. They do not authorize choosing a different SHTAR definition. Compiling a contract must not silently replace its model, interpolation rule, discretization, random stream, exercise policy or sampler.

These state-preservation rules do not prohibit mathematical simplification inside a registered model valuation. Evaluating a child's value function instead of expanding its future payments changes how its value is calculated, while leaving its SHTAR state, update rules and accounting obligations intact. It is distinct from substituting a different product-state definition.

10.4 Evidence

Compilers should retain event traces and test vectors comparing source meaning and declared state updates with target behavior. Useful vectors exercise equality boundaries, missing data, changed state types, shared contexts, duplicate events, transfers and unresolved decisions.

Compiler and numerical-profile choices are recorded separately. Different numerical approximations can be supported profiles, but are not automatically bitwise-identical translations of one abstract contract.

11 Valuation, estimators and hedge interpretation

The valuation interpretation establishes the context's direct or synthetic value status and binds the observations and model computations needed to value the defined claims. Either status can require a model. This chapter specifies a generic valuation-profile boundary, separately from operational event processing and implementation lowering. It adds no Monte Carlo operations to the contract AST and does not require a PDE provider to implement regression.

11.1 Valuation conventions

A profile identifies valuation time, the direct asset used as reporting numeraire, measure and intertemporal conversion, common output units, ownership, claim conventions and supported aggregation. It distinguishes future quantities, exchange-value observations, model-dependent conditional values and numerical contributions at the valuation frame, separately from direct or synthetic status. Required direct conversions can be supplied, derived or model-computed under the accepted conventions; no currency asset or complete pairwise exchange table is required.

Valuation of the remaining product at time t concerns residual rights and obligations in the selected economic ownership scope, including recognized unpaid claims due before t and future contingent claims. Settled historical payments are excluded from those remaining obligations, while any surviving contractual dependence on their records is retained in state. A delivered child already owned outside that scope is not automatically part of the parent's remaining value.

The profile may convert and aggregate dated residual payments numerically without rewriting their contractual dates or assets. Discounted valuation does not imply actual reinvestment of earlier receipts. A terminal-wealth or total-return interpretation must additionally specify its holding, reinvestment and financing rules; there is no uniquely determined terminal amount for already settled payments in their absence.

Choosing an output denomination does not by itself define a stochastic numeraire process or a pricing measure. A profile using a numeraire-based measure must supply the relevant process, gain and settlement conventions and satisfy that method's mathematical conditions. Converting a model-computed quantity or publishing it as a context value preserves its model provenance and does not itself establish or remove direct status; the acceptance basis remains separately bound.

Monte Carlo aggregation retains the original population, including zero additional contributions from a terminated parent while preserving any outstanding child claims in scope. Every simulated continuation starts from the accepted historical sufficient state and updates only future observations; historical settled payments and consumed fixings are not posted again. Context values contribute only when the economic ownership graph makes them part of a claim.

11.2 Control variates

Let Y be a numerical payoff contribution and let Z_j have registered known expectations μ_j, in compatible valuation units:

$$C_j = Z_j - \mu_j, \qquad \widetilde{Y} = Y - \sum_j \widehat{\beta}_j C_j.$$

The estimator policy identifies fixed, independently trained or cross-fitted coefficients. It records known expectations, their observable or model provenance, sample grouping, scaling, rank handling, training information and sampling-error conventions. Knowing an expectation analytically under the model does not determine the control's direct or synthetic status. An accepted market exchange value is a valid

control expectation only when the registered valuation conventions establish the matching expectation; direct status alone is insufficient. A sample mean is not silently substituted for a known expectation.

Raw values and diagnostics remain available separately. Sampling error does not establish absence of discretization, model or calibration error.

11.3 Hedge interpretation

If each Z_j is a defined tradable instrument's numerical payoff and μ_j is its consistently valued current exchange value in the same reporting units, $\widehat{\beta}_j$ is fitted exposure in instrument units. The offsetting static position is $-\widehat{\beta}_j$. For a hedge priced from an accepted direct exchange value, the registered expectation and accepted value must agree under the selected valuation conventions. Model-dependent exposure estimates must be identified as such under either value status; direct status alone does not establish an executable hedge.

One application profile uses discounted forward payoffs under compatible deterministic carry:

$$C_j = D(0,t_j)(S_{t_j} - F(0,t_j)).$$

Here S and F are exchange values in that profile's chosen units, and D supplies its intertemporal conversion. This formula is a profile example, not a currency-special rule or a general formula for arbitrary numeraires. The discount is already included; coefficient units follow the payoff multiplier. Synthetic-state controls can reduce variance without being tradable hedges. Neither interpretation is automatically a dynamic hedge or spot sensitivity.

Collinearity may prevent unique hedge quantities. Scaling and solution conventions must be disclosed. Slopes fitted on normalized controls must be converted back to original instrument units when reported as hedges.

SHTAR

11.4 Continued products

These rules also apply to delivered products. Observed, derived or model-computed child values eventually yield numerical contributions under the selected reporting numeraire. Closed-form valuation and conditional numerical continuation are computational choices available under either direct or synthetic status, subject to registration. The child need not share its parent's state schema or use an independent market realization.

For example, when a SHTAR swaption delivers a separately defined swap, a model with a registered swap-value function can evaluate it immediately at that frame from the current model state and the required swap terms and state. It need not expand each future swap payment to obtain that value. A model without such a function may instead use a supported PDE, branching or conditional MC, or another valuation method. Both routes value the same delivered swap; neither changes its declared state or cancels its future accounting obligations. The conditioning and exclusive-counting rules of Section 8 still apply.

The SPX application overview explains the product and valuation context at a high level. Numerical implementation choices are separate from language semantics.

12 Conformance, errors and extensions

12.1 Validation levels

Structural validation checks version, required artifact metadata, fields, operators and JSON types. The supplied schema/validator performs this level. Appendix A specifies the artifact-metadata and attribution requirements for documents, compilers and reports.

Semantic linking checks names, key signatures, expression/state types, units, scope, context no-payment effects, phases, cycles, capabilities and constructors.

Registration/runtime validation checks actual bindings, information availability, historical sufficient state, settlement evidence and residual-claim scope, value-status acceptance conditions and computational provenance, ownership, successor state, dates, decisions, multiplicity and transaction invariants.

Schema validity alone does not establish a valid security, conforming compiler or priceable deal.

12.2 Failure classes

Implementations distinguish malformed syntax, unsupported version/operator, unresolved import, invalid type/unit, missing or ambiguous key, unavailable history, future-information access, invalid context effects, dependency cycles, conflicting events, invalid transfers/decisions, nonadvancing wakeups, unsupported dates/horizons/routes, numerical-domain failure and resource exhaustion.

An error never becomes a zero payment or false condition. Diagnostics identify the definition, instance/event where available, failing expression/key and affected capability.

12.3 Extensions

Extensions have namespaced identities, versions, types and semantics. Unsupported required extensions are rejected. Descriptive metadata cannot change execution.

Extensions may add pure functions, transport types, frontends, event sources or methods. They cannot silently redefine identity, contexts, deliveries, decisions or numerical leaves.

12.4 Adversarial cases

Conformance checks must cover:

1. Different successor types on realized branches.
2. A separately defined security delivered and valued through exactly one route.
3. Two readers of one post-event context and only one compatible update.
4. Equal values from distinct producers remaining distinct for a many-valued key.
5. An unresolved decision posting no alternative's payment.
6. A terminated parent with an outstanding deferred child claim.
7. A valid returned date outside provider support.
8. A replacement attempting to change an unchanged key's result type.
9. A repeated event attempting a duplicate delivery.
10. Generated code or output dropping or altering required artifact metadata (Appendix A).

SHTAR

11. A private model-produced context value declared direct without an accepted market-agreement basis, or a valid direct value incorrectly declared synthetic merely because computing it uses a model.
12. An immediate-delivery exchange value substituted for a different settlement or obligor claim.
13. A share- or bond-denominated result incorrectly treated as requiring currency output, or a derived direct conversion incorrectly requiring a stored pairwise quote.
14. A settled historical payment posted again or silently carried into a later terminal payment without a specified strategy.
15. A past-due unpaid claim discarded solely because its due date precedes valuation.
16. A reporting-numeraire change incorrectly replacing the contractual denomination or deliverable asset.
17. Historical Asian fixings reapplied to the initial sum/count, or one accepted fixing counted twice through replay or shared readers.
18. A finite contractual fixing schedule replaced by a simulation grid, or a continuous-average quadrature presented as an exact contractual fixing rule.
19. An authored aggregate definition losing information needed by a remaining condition or export, or erasing audit records needed for settlement or reconciliation.
20. Conditional missed-coupon memory treated as an already due claim, or heterogeneous coupons aggregated while losing their distinct eligibility, settlement, accrual or survival rules.
21. An interpreter, pricer or compiler replacing a declared coupon or fixing list with an aggregate, despite producing the same financial deliveries; an alternative representation requires an explicitly selected SHTAR definition.
22. Direct status retained after its agreement or liquidity basis ceases, or an unsupported solver silently selected when a previously direct bond or option becomes indirect.
23. A continuing review omitting its next observation request, or epsilon reusing the current event or accepting an unordered timestamp tie.
24. A calendar date assigned an implicit midnight/time zone, or a timestamp converted to numerical model time inside the product definition instead of by the provider.
25. Epsilon replaced by an arbitrary fixed step or used to claim that coarse monitoring exactly reproduces a continuous barrier without a declared supported interpretation.

Unsupported states require explicit rejection or re-registration/reconciliation, not reinterpretation.

13 Current implementation and next ports

The Python implementation supports an application subset: arbitrary Python state, returned profile dates, child fixed claims, saved-calibration observations and numerical regression reports. It does not yet implement the proposed IR, general typed-context registry, complete dependency coordinator or general AI/compiler backend. The generic asset, chronological-ledger, residual-claim, averaging and timestamp/epsilon semantics do not expand the existing SPX/USD adapter's executable scope or assert an implemented general settlement ledger, Asian pricer or intraday/tick scheduler.

SHTAR supports Monte Carlo valuation for selected products with exercise rights.

The provider overview and SPX introduction describe the application at a high level. The SPX application PDF explains the economic purpose and workflow.

A Framework and artifact metadata

The public publication profile identifies the SHTAR framework separately from a deal's economic terms, observations, state and payoff rules. Its attribution member is:

```
{
  "attribution": {
    "framework": "SHTAR",
    "credit": "SHTAR"
  }
}
```

A.1 Machine-readable artifacts

The public schema uses the identifier `urn:sdl:syntax:0.1-draft:public`. Frontends and compilers preserve the framework metadata in the checked representation and generated artifacts. Attribution is not a contractual parameter or valuation input. Private source-artifact metadata is managed by the producing application and is not included in this public profile.

A.2 Human-facing documents and reports

A document or report may identify SHTAR in its title, footer or appendix. Introductions and deal economic terms focus on the contract, examples and results.

SHTAR

A.3 Validation and artifact identity

The public schema requires the framework and credit values shown above. Structural validation rejects missing fields or changed framework values. Artifact digests include the metadata alongside the semantic document. These metadata rules do not change the financial meaning of a product or introduce a licensing mechanism.