



UNDERSTAND THE EVIDENCE.

Validation Guide

The `lumicron_validate` Python package

Edition	October 2026
Documentation	6 October 2026
Publisher	Elyon Semiconductor

Lumicron Validation Guide

© 2026 Elyon Semiconductor. All rights reserved.

Documentation for the Python API bundled with Lumicron.

Chapters, examples, figures and API signatures are preserved from the current application guide sources.

Web edition: <https://piclumicron.com/docs/>.

Use the documentation matching your installed release.

Table of contents

Lumicron Validation API	1
1 Conventional DRC	2
1.1 Named DRC checks in Python	3
1.2 Explicit region decks (schema 2)	3
2 Semantic DFX	5
2.1 Extension boundary	6
3 Results and applicability	7
3.1 Saved native DRC reports	7
Appendices	9
API reference	9

Lumicron Validation API

Use `lumicron_validate` to author fabrication rules and evaluate semantic design requirements.

```
import lumicron_validate as lv
```

DRC checks physical geometry and layers: width, spacing, area and enclosure. **DFX** checks design requirements that depend on what an object is, such as the orientation of an explicitly addressed instance's optical port. These are different evidence scopes.

The packages ship together in Lumicron's bundled Python runtime. This guide does not imply a separately published PyPI distribution. Run examples with that runtime or an environment containing all three bundled packages and their dependencies.

Task	Import	Guide
Design a PIC, place and route instances	<code>import lumicron as lm</code>	Layout API
Author layers, PCells and PDK packages	<code>import lumicron_pdk as lpdk</code>	PDK Development API
Author DRC and semantic requirements	<code>import lumicron_validate as lv</code>	This guide

The native DRC/LVS session's binding and applicability rules are unchanged. A Python check result is evidence only for the inputs and scope it actually evaluated.

Reusable rule code can also be a two-file Lumicron engineering package: `lumicron.toml` plus `checks.py`, with `validation = "checks.py"` under `capabilities`. A declared process dependency describes the provider's applicability; it does not implicitly run or combine its rules. See [Examples/Packages/SimpleValidation](#) and the Layout guide's *Packages and project environments* chapter. Its semantic orientation check uses an authored instance and named port, not polygon proximity. The neighboring `GratingTools` example shows that ordinary utilities use the same package model.

CHAPTER 1

Conventional DRC

A deck contains named rules with stable IDs, explicit units and severity. Existing rule constructors retain their syntax and produce the same version-1 deck JSON. They describe rules; constructing a deck does not execute or qualify them.

```
import lumicron_pdk as lpdk
import lumicron_validate as lv

CORE = lpdk.Layer(1, 0)
DRC = lv.drc.Deck(name="Example process", rules=[
    lv.drc.min_width(CORE, 0.2, id="core.width", name="Core minimum width"),
    lv.drc.min_spacing(CORE, 0.3, id="core.spacing", name="Core isolation"),
])
DRC.write("example.lumdrc")
```

Export this object as DRC in a PDK package to embed it during `lpdk.pack_pdk`. An existing `drc.json` remains readable; conflicting Python and JSON declarations are rejected. Layer definition and stippling belong to `lumicron_pdk`.

`min_width` uses projection metrics in micrometres. `min_spacing` describes Euclidean isolation between disconnected masks. `min_area` uses square micrometres. `enclosure` takes inner and outer layers. Supported grids, geometry and engine limits are checked when the native DRC engine executes the deck. See the generated reference for signatures.

The existing polygon worker and authored LVS comparator are now owned by this package. Their native request/result contracts and evidence scopes remain unchanged. Optional external DRC/LVS integrations are not bundled engines and their absence is not a successful check.

1.1 Named DRC checks in Python

Export DRC from the PDK's package root. The packer generates the existing schema-v1 `drc.json`; the native checker reads that data and does not execute Python rules.

```
from lumicron_validate import drc

DRC = drc.Deck(name="Example development rules", rules=[
    drc.min_width(LAYER.CORE, threshold_um=0.45,
        id="CORE.W.1", name="Core minimum width", metrics="projection"),
    drc.min_spacing(LAYER.CORE, threshold_um=0.5,
        id="CORE.S.1", name="Core isolation spacing"),
    drc.min_area(LAYER.CORE, threshold_um2=0.2,
        id="CORE.A.1", name="Core minimum area", severity="warning"),
    drc.enclosure(LAYER.CORE, outer_layer=LAYER.CLAD, threshold_um=0.1,
        id="CLAD.E.1", name="Cladding enclosure",
        description="Illustrative threshold; replace with process requirements."),
])
```

Linear thresholds are micrometers; area thresholds are square micrometers. Each rule needs a unique stable id and a human-readable name. All constructors accept `severity="error"`, `"warning"`, or `"info"`, plus an optional description. Identifiers use ASCII letters, digits, underscore, dot, colon, or hyphen. Decks support 1–4096 rules and a maximum encoded size of 4 MiB. Thresholds must be positive and finite, at most 1,000,000 μm or 1,000,000,000,000 μm^2 .

`drc.min_width` currently supports projection metrics only. Spacing checks isolation between disconnected masks; area checks merged filled regions; enclosure checks containment and clearance. The constructors do not expand the native engine's supported geometry or qualification. Grid representability and supported geometry are still checked when DRC runs. There is no radius rule in this deck version.

`DRC.write("rules.lumdrc")` atomically writes a standalone deck; `DRC.to_dict()` returns an independent JSON-compatible projection. A package may retain its existing handwritten `drc.json`. If it also exports DRC, both decoded decks must be equal; otherwise packaging fails. Python deck layers must exist in the exported LAYER.

1.2 Explicit region decks (schema 2)

Standalone schema-2 `.lumdr` decks express additional region operations without changing the meaning of schema-1 `min_width`, `min_spacing`, `min_area` or `enclosure`. The Python constructors above continue to author schema 1. Schema 2 currently requires every rule to use `check: "region"`; mixed-schema execution is rejected. This is an engineering preview, not fabrication signoff.

```

{
  "drc_version": 2,
  "deck_name": "Explicit edge checks",
  "rules": [{
    "id": "CORE.W.1", "name": "Core width", "severity": "error",
    "check": "region", "layer": [3, 0],
    "region": {
      "operation": "edge_width", "threshold_um": "0.199",
      "angle": 45, "discardZeroArea": false
    }
  }]
}

```

Distances and areas in region specifications are positive decimal strings in μm and μm^2 , converted exactly to the input DBU. Threshold representability is checked before execution. Margins must be stated in the deck's actual threshold; no implicit global fabrication margin is applied.

Operation	Required region fields (besides operation)
edge_width, edge_space	threshold_um, angle (45 or 90), discardZeroArea
edge_pair_filter	threshold_um, angle, space, area_um2, perimeter_um, combine (and or or)
area_not_equal	area_um2
off_grid	grid_um
inside_sized_raw	otherLayer ("layer/datatype"), size_um
overlapping	otherLayer ("layer/datatype")

edge_space includes connected notches. Edge checks use Euclidean clipping and transparent shielding. Compound conditions select and combine edge-pair polygons. inside_sized_raw selects entire inside shapes against individually expanded raw outer polygons; relevant oblique outer sizing is unsupported. overlapping selects whole shapes with positive-area overlap, excluding boundary-only touch. area_not_equal tests each merged polygon; an absent outline is not invented as a finding. off_grid examines all input layer/datatype pairs (its rule layer field is a placeholder) and reports off-grid vertices.

The current native path retains bounded hierarchy expansion and materialization limits. Unsupported geometry or exhausted limits fails the whole run; it is not a clean result. Package rule declarations are not automatically upgraded to this deck schema. Select the intended standalone deck explicitly.

CHAPTER 2

Semantic DFX

A semantic check uses a real authored identity, not a guessed component name or nearby polygon. The initial bounded check asks whether one instance port faces an expected direction in its parent cell.

```
import lumicron_validate as lv

result = lv.dfx.port_orientation(
    layout,
    parent_cell="TOP",
    instance_id=authored_instance_id,
    port="fiber",
    expected_degrees=90,
    tolerance_degrees=0.01,
)
print(result.status, result.reason)
print(result.is_current(layout))
```

Obtain `authored_instance_id` from the authored `CellRef.authored_id` in the Layout snapshot or its validated semantic projection. Two identical placements have different IDs. The check resolves that exact ID and named port; it does not search geometry for a replacement.

The check applies the existing reflection-then-rotation convention to the child port. Translation and positive magnification do not change direction. The frame is the named parent definition, not the top-level world frame. A missing or ambiguous ID, legacy identity, missing port, or array reference is **unsupported**. Array-occurrence and arbitrary nested-world addressing are not established by this check.

A failed angular comparison is **failed**, not unsupported. Settings are retained in the result, including the angular tolerance. Directions use degrees modulo 360 and the shortest angular difference.

2.1 Extension boundary

New semantic checks should follow the same rule: address exact authored evidence, identify its coordinate frame, record settings and bind the result to its inputs. Add a narrowly owned function and independent tests. No generic graph, implicit component matching, or blanket DFX qualification is provided here.

CHAPTER 3

Results and applicability

`ValidationResult` records the check, status, reason, exact serialized Layout digest, authored address, expected/tolerance/actual direction and coordinate frame. `to_dict()` is a machine-readable projection.

`is_current(layout)` compares the current serialized Layout inputs with the evaluated snapshot. Moving, deleting or editing an instance makes an earlier result stale even when that individual orientation would still pass if recomputed. The result does not authorize native DRC/LVS to show a current pass.

Digest equality establishes local input association. It is not cryptographic authenticity. Runtime-only data excluded from the Layout serialization is outside this check's scope.

`lumicron_validate.testing` retains the existing Layout assertions and pytest plugin. Assertions raise on failure and do not create persistent qualification records. `lumicron_validate.lvs` retains the authored LVS comparison implementation and its supported/unsupported boundaries.

The old package is a deprecated compatibility bridge for existing projects. New code uses the three canonical packages; compatibility names are not separate implementations.

3.1 Saved native DRC reports

The DRC report menu offers **Save Report...**, **Open Report...**, **Export JSON...**, and **Export Human-Readable Report...**. The full report window uses the same result as the Layout bottom panel. Saving includes every retained finding, including findings beyond the panel's 200-row preview.

A `.lumdrcreport` is a versioned JSON artifact containing the typed result, application version, source and canonical-layout digests, selected top, DBU, deck/PDK revisions, semantic/source evidence, exact integer finding geometry, measurements and completion/truncation status. JSON

export uses the same format. The content digest detects accidental changes; it is not a signature or foundry certification. `.lumdrc` remains the rule-deck extension.

Opening a report does not run DRC. **Matched** means its bound layout, top, deck and associated inputs match current evidence, permitting overlays. **Stale** and **Unbound** reports remain readable but cannot project current overlays. A malformed, unsupported or corrupt report is rejected. Opening historical evidence does not publish a new verification run or restore its old session.

The searchable HTML export can be read without Lumicron. It lists counts, measurements, geometry and source/deck context. It is a presentation of the same report model, not an alternative import format. Supported report limits remain 10,000 retained findings, the typed result's aggregate bounds, and 64 MiB encoded bytes. Explicitly truncated or failed runs never become complete reports.

Finding fingerprints support exact evidence comparisons. They include rule revisions and geometry; their equality does not prove continuity through arbitrary source, hierarchy or deck changes. No fuzzy cross-revision matching is performed.

API reference

Signatures below are generated from the bundled Lumicron API. Use the task-oriented chapters for examples and constraints. Builder class names describe return values; users normally obtain them through CELL actions and chained methods.

lv.ValidationResult

One bounded check, bound to exact serialized Layout inputs and settings.

```
lv.ValidationResult(check: str, status: str, reason: str, input_digest: str,  
    ↪ parent_cell: str, instance_id: str, port: str, expected_degrees: float,  
    ↪ tolerance_degrees: float, actual_degrees: float | None = None, frame: str =  
    ↪ 'parent-cell') -> None
```

lv.dfx

Bound semantic checks on explicitly addressed authored design objects.

lv.drc

Author named DRC rules in Python; package or write the existing v1 JSON deck.

lv.lvs

Revision-bound LVS for authored PCell instances and explicit routed ports.

lv.open_validation_guide

Open the bundled Validation API guide in the default PDF viewer.

```
lv.open_validation_guide() -> pathlib.Path
```

lv.testing

Layout assertions shipped in Lumicron's bundled runtime.

lv.validation_guide_path

Filesystem path to the bundled Validation API guide.

```
lv.validation_guide_path() -> pathlib.Path
```

lv.drc.Deck

A validated v1 deck; export it as PDK.DRC or write a standalone .lumdrc.

```
lv.drc.Deck(*, name: 'str', rules: 'Sequence[_Rule]', description: 'str | None' = None)
```

lv.drc.min_width

Describe minimum projection width in μm ; metrics must be 'projection'.

```
lv.drc.min_width(layer: 'LayerSpec', threshold_um: 'float', *, id: 'str', name: 'str',  
    ↪ metrics: 'str' = 'projection', severity: 'str' = 'error', description: 'str |  
    ↪ None' = None) -> '_Rule'
```

lv.drc.min_spacing

Describe Euclidean isolation spacing between disconnected masks in μm .

```
lv.drc.min_spacing(layer: 'LayerSpec', threshold_um: 'float', *, id: 'str', name:
↳ 'str', severity: 'str' = 'error', description: 'str | None' = None) -> '_Rule'
```

lv.drc.min_area

Describe minimum filled area of a merged mask region in μm^2 .

```
lv.drc.min_area(layer: 'LayerSpec', threshold_um2: 'float', *, id: 'str', name: 'str',
↳ severity: 'str' = 'error', description: 'str | None' = None) -> '_Rule'
```

lv.drc.enclosure

Require outer_layer to contain layer with at least threshold_um clearance.

```
lv.drc.enclosure(layer: 'LayerSpec', outer_layer: 'LayerSpec', threshold_um: 'float',
↳ *, id: 'str', name: 'str', severity: 'str' = 'error', description: 'str | None' =
↳ None) -> '_Rule'
```

lv.dfx.ValidationResult

One bounded check, bound to exact serialized Layout inputs and settings.

```
lv.dfx.ValidationResult(check: str, status: str, reason: str, input_digest: str,
↳ parent_cell: str, instance_id: str, port: str, expected_degrees: float,
↳ tolerance_degrees: float, actual_degrees: float | None = None, frame: str =
↳ 'parent-cell') -> None
```

lv.dfx.port_orientation

Check an authored instance port's direction in its parent-cell frame.

```
lv.dfx.port_orientation(layout: lumicon.models.layout.Layout, *, parent_cell: str,
↳ instance_id: str, port: str, expected_degrees: float, tolerance_degrees: float =
↳ 1e-06) -> lumicon_validate.dfx.ValidationResult
```

lv.testing.assert_cell_unchanged

Compare the layout's geometry signature against a stored snapshot at tests/_snapshots/<key>.json. Failure means the cell's geometry drifted since the snapshot was recorded.

```
lv.testing.assert_cell_unchanged(target: 'Any', *, snapshot_key: 'str | None' = None,  
    ↪ tolerance: 'float' = 1e-06) -> 'None'
```

lv.testing.assert_no_overlap

Fail if any two polygons on layer overlap (positive area intersection). Uses shapely's unary_union — if the sum of individual areas exceeds the union area by more than a tiny epsilon, there's an overlap, and we extract the intersection regions for the failure overlay.

```
lv.testing.assert_no_overlap(target: 'Any', layer: 'Any') -> 'None'
```

lv.testing.assert_route_lengths_equal

Assert every route in routes has the same centerline length within tol μm . Lifts the shortest + longest centerlines into the offending-geometry payload so the failure overlay can show which routes are the outliers.

```
lv.testing.assert_route_lengths_equal(routes: 'Iterable[Any]', *, tol: 'float' = 0.01,  
    ↪ units: 'str' = ' $\mu\text{m}$ ') -> 'None'
```

lv.testing.assert_ports_connected

Fail if any port in the layout is dangling (not part of any extracted net). Backed by lm.extract_netlist + the PortInfo .net_id is None orphan marker the netlist subsystem already surfaces.

```
lv.testing.assert_ports_connected(target: 'Any') -> 'None'
```

lv.testing.assert_netlist_matches_schematic

Run LVS between the layout's extracted netlist and a .lumsch file. Wraps lumicron_lvs.compare_netlists. Fails if any component or net diff is reported.

```
lv.testing.assert_netlist_matches_schematic(target: 'Any', schematic_path: 'str |  
↳ Path') -> 'None'
```

lv.testing.LumicronAssertionError

Failure exception raised by every assertion in this package.

```
lv.testing.LumicronAssertionError(message: 'str', offending: 'OffendingGeometry')
```

lv.testing.OffendingGeometry

The structured failure payload attached to every LumicronAssertionError. Pytest's pytest_runtest_makereport hook (in _plugin.py) lifts this off the exception and emits it in the JSON event stream the Mac Tests tab consumes.

```
lv.testing.OffendingGeometry(*, assertion: str, message: str, polygons:  
↳ list[lumicron_validate.testing._types.OffendingPolygon] = <factory>, ports:  
↳ list[lumicron_validate.testing._types.OffendingPort] = <factory>, source_loc:  
↳ lumicron_validate.testing._types.SourceLoc | None = None) -> None
```

lv.testing.OffendingPolygon

One polygon to highlight in red on the viewer canvas.

```
lv.testing.OffendingPolygon(*, points: list[tuple[float, float]], layer_name: str,  
↳ color: str | None = None) -> None
```

lv.testing.OffendingPort

One port marker — used by assert_ports_connected so the viewer can drop a circle on each dangling port instead of drawing a polygon.

```
lv.testing.OffendingPort(*, x: float, y: float, layer_name: str, direction: float =
↳ 0.0) -> None
```

lv.testing.SourceLoc

Where the failing assertion was called. Lets the Mac side pop the editor on the asserting line so the user can read the assertion alongside the geometry. lumicon.gui_bridge already has a SourceLoc — we keep this one separate for the sake of test-package isolation, but the field shape matches.

```
lv.testing.SourceLoc(*, file: str, line: int, function: str | None = None) -> None
```

lv.lvs.UnsupportedLVS

There is not enough supported evidence to claim a comparison result.

lv.lvs.compare_authored_layout

Compare immutable captured design and independently exported geometry facts.

```
lv.lvs.compare_authored_layout(manifest, expected, *, layout_sha256, schematic_sha256)
```

lv.ValidationResult.is_current

Whether the evaluated serialized Layout is still exactly present.

```
lv.ValidationResult.is_current(layout: lumicon.models.layout.Layout) -> bool
```

lv.ValidationResult.to_dict

Machine-readable result including its evidence and rule settings.

```
lv.ValidationResult.to_dict() -> dict
```

lv.drc.Deck.to_dict

Return an independent JSON-compatible schema-v1 deck projection.

```
lv.drc.Deck.to_dict() -> 'dict'
```

lv.drc.Deck.write

Atomically publish this deck as JSON, typically to a .lumdrc file.

```
lv.drc.Deck.write(path: 'str | Path') -> 'Path'
```

lv.dfx.ValidationResult.is_current

Whether the evaluated serialized Layout is still exactly present.

```
lv.dfx.ValidationResult.is_current(layout: lumicron.models.layout.Layout) -> bool
```

lv.dfx.ValidationResult.to_dict

Machine-readable result including its evidence and rule settings.

```
lv.dfx.ValidationResult.to_dict() -> dict
```