# The lpmresonance Package
Documentation

Prajwal Dhondiram Udanshive

Draft compiled on November 19, 2025

**Abstract**

This working draft describes the lpmresonance package as it would appear in a CTAN distribution. It combines user-facing guidance, implementation commentary, and step-by-step explanations of how the Python–LaTeX bridge orchestrates lattice-path graphics. Code listings rely on the minted environment, and all diagrammatic examples are generated with the actual package macros to keep the documentation faithful to the repository.

## Contents

**1 Orientation** **2**
1.1 Audience . . . 2

**2 Repository at a Glance** **2**

**3 Toolchain and Compilation** **2**
3.1 Dependencies . . . 2
3.2 Editable installation . . . 3
3.3 Compiling with PythonTeX and minted . . . 3

**4 Quick Start Recap** **3**

**5 Mechanism: from bit string to TikZ picture** **4**
5.1 Six stages at a glance . . . 4
5.2 TeX–Python bridge . . . 4
5.3 Geometry construction . . . 5
5.4 Emitter output . . . 6
5.5 Drawing layer . . . 7

**6 Macro reference** **8**
6.1 Declaring and referencing paths . . . 8
6.2 Schubert picture helpers . . . 8

**7 Python module details** **8**
7.1 Between-region construction . . . 8
7.2 Cache fencing . . . 9
7.3 Manifest serialization and hashing . . . 9

**8 Caching and housekeeping** **10**

**9 Worked diagrams** **10**
9.1 Single annotated path . . . 10
9.2 Between-region shading . . . 11

10 Testing, troubleshooting, and release checklist 11

11 Future refinements 12

# 1 Orientation

lpmresonance couples TikZ drawing helpers with a Python back-end that converts bit strings into cached coordinates. The package generates annotated lattice paths and between-region shadings from binary strings.

Unlike monolithic TikZ solutions, lpmresonance defers the heavy lifting to PythonTeX: the first compilation pass runs the Python helper, emits JSON & TEX artifacts in `lp-cache/`, and later passes pick up the generated coordinates without recomputation.

## 1.1 Audience

This document serves three overlapping roles:

- A quick reference for mathematicians who only need the high-level macros.
- A detailed guide for maintainers who want to adjust the Python emitter or the TikZ front-end.
- A reproducible tour for CTAN reviewers who expect explicit build instructions, dependency disclosures, and implementation notes.

# 2 Repository at a Glance

| Path | Contents |
|---|---|
| `tex/latex/lpmres` | Package source: `.sty` dispatcher and modular `.code.tex` files for base helpers, Python bridge, lattice-path drawing, between regions, grids, and picture environments. |
| `python/lpm_paths` | Python modules invoked by PythonTeX: API surface, geometry types, caching, hashing, manifest helpers, and TeX emitter routines. |
| `docs/` | MkDocs site used during development. This LATEX document complements it with a CTAN-flavored narrative. |
| `examples/` | Ready-to-compile demonstrations plus cached outputs generated during testing. |
| `scripts/` | Maintenance tasks (`clean-cache.sh`, `build-docs.sh`, regression helpers). |
| `tests/` | Python unit tests for geometry and manifest serialization. |

Table 1: Top-level folders relevant to end users (left) and developers (right).

# 3 Toolchain and Compilation

## 3.1 Dependencies

- Python 3.9–3.13 with the `lpm_paths` package installed in the active virtual environment.
- TEX Live 2022 or newer with PythonTeX, TikZ, and minted (Pygments) available.

- `latexmk` (or any equivalent driver) configured with `-shell-escape` so that both PythonTeX and minted may spawn helper processes.

### 3.2 Editable installation

```
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -e .
```

Listing 1: Editable install keeps `lpm_paths` synchronized with the working tree.

### 3.3 Compiling with PythonTeX and minted

Either adapt `examples/latexmkrc` or pass the relevant knobs manually:

```
latexmk -pdf -shell-escape \
  -interaction=nonstopmode \
  lpmresonance-doc.tex
```

Listing 2: The `-shell-escape` flag is mandatory for both PythonTeX and minted.

The first run executes PythonTeX cells embedded in `\lpDeclarePath` / `\shadeBetweenBits` and produces cached coordinates. Subsequent runs reuse those artifacts until the bit strings or sanitized names change.

## 4 Quick Start Recap

A minimal document resembles the MkDocs walkthrough yet highlights minted listings inside the instructions.

```
\documentclass{article}
\usepackage{lpmresonance}
\begin{document}
\lpDeclarePath{demo}{01011010}
\begin{schubertpic}
  \drawGrid{demo}
  \drawLatticePath[
    lplpath/label upmarks,
    lplpath/show inside corners,
  ]{demo}
\end{schubertpic}
\end{document}
```

Listing 3: Hello-world lattice path with automatic grid and annotations.

Running `latexmk -pdf -shell-escape demo.tex` creates `lp-cache/path-demo-*.tex/json`, then draws the path on the second pass.

# 5 Mechanism: from bit string to TikZ picture

## 5.1 Six stages at a glance

**Stage 1: Declaration in TEX** — `\lpDeclarePath` collects a human-readable name and a bit string.

**Stage 2: PythonTeX payload** — the macro injects a short Python snippet that calls `lpm_paths.api.declare_path`

**Stage 3: Geometry build** — `LatticePath.from_bits` converts `0/1` steps into coordinates, upmarks, and inside-corner metadata.

**Stage 4: Cache emission** — the `TeXEmitter` writes `lp-cache/path-*.tex` (macro definitions) and `lp-cache/path-*.json` (manifest for debugging).

**Stage 5: Drawing layer** — `\drawLatticePath`, `\shadeBetween`, and related commands consult the cached macros.

**Stage 6: Optional shading helpers** — `\shadeBetweenBits` feeds two bit strings into `between_polygon` and records the resulting polygon.

## 5.2 TeX–Python bridge

`tex/latex/lpmres/lpmres-python.code.tex` contains the high-level bridge:

```
\newcommand\lpDeclarePath[2]{%
  \pyc{import json;
      from lpm_paths import declare_path_from_json;
      spec = {"name": r"""#1""", "bits": r"""#2"""};
      print(declare_path_from_json(json.dumps(spec, ensure_ascii=False)))}%
  \lp@inputifready{lp@lastdeclaredpathfile}%
}
\newcommand\shadeBetweenBits[4]{%
  \pyc{import json;
      from lpm_paths import between_from_json;
      spec = {"L": r"""#1""", "U": r"""#2""",
              "lname": r"""#3""", "uname": r"""#4"""};
      print(between_from_json(json.dumps(spec, ensure_ascii=False)))}%
  \lp@inputifready{lp@lastdeclaredbetweenfile}%
  \lp@ensurebetweenplaceholder{#3}{#4}%
}
```

Listing 4: PythonTeX shims for declaration and shading (tex/latex/lpmres/lpmres-python.code.tex).

Each macro emits fresh TEX code whenever the Python helper prints path metadata. The helper itself lives in `python/lpm_paths/api.py`:

```python
def declare_path_from_json(spec_json: str) -> str:
    spec = json.loads(spec_json)
    bits = spec.get("bits")
    name = spec.get("name")
    cache_id = spec.get("cache_id")
    emitter = TeXEmitter(Cache.make())
    g1, g2, g3 = emitter.write_path(bits=bits, name=name, cache_id=cache_id)
    return "\n".join([g1, g2, g3])
```

Listing 5: `declare_path_from_json` entry point (python/lpm_paths/api.py).

### 5.3 Geometry construction

The `LatticePath` dataclass is the canonical representation of a path.

```python
@dataclass(frozen=True)
class LatticePath:
    bits: str
    coords: List[Coord]
    upmarks: List[Upmark]
    corners: List[int]
    insideCorners: List[int]
    ellmap: Dict[int, int]

    @staticmethod
    def from_bits(bits: str) -> "LatticePath":
        x = y = 0
        coords = [(0, 0)]
        upmarks, corners, insideCorners = [], [], []
        prev = None
        stepIndex = 0
        for b in bits:
            stepIndex += 1
            if b == "0":
                x += 1; cur = "E"
            else:
                y += 1; cur = "N"; upmarks.append(stepIndex)
            coords.append((x, y))
            if prev is not None and prev != cur:
                corners.append(stepIndex - 1)
            if prev == "E" and cur == "N":
                insideCorners.append(stepIndex - 1)
            prev = cur
        ...
        return LatticePath(bits, coords, upmarks, corners, insideCorners, ellmap)
```

Listing 6: Core geometry builder (python/lpm_paths/types.py).

This routine also computes `ellmap` entries (maximal East reach on each horizontal level) so TikZ styles can align grids and axes.

### 5.4 Emitter output

```python
class TeXEmitter:
    def write_path(self, bits: str, name: str, cache_id: str | None = None):
        safe = _sanitize_name(name)
        payload = {"op": "declare_path", "bits": bits, "name": name,
                   "ver": EMITTER_VERSION, "cache_id": cache_id or ""}
        key = key_of(payload)
        texname = f"path-{safe}-{key}.tex"
        jsonname = f"path-{safe}-{key}.json"
        texpath = self.cache.file(texname)
        lp = LatticePath.from_bits(bits)
        body = ["\\makeatletter"]
        body.append(f"\\expandafter\\gdef\\csname
        ↪  lp@path@coords@{safe}\\endcsname"
                    f"{{{_formatCoords(lp.coords)}}}")
        ...
        atomic_write(texpath, "\n".join(body) + "\n")
        warn = self._safe_name_warning("path", safe, name)
        g1 = "\\makeatletter\n" + _gdef(f"lp@pathfile@{safe}",
        ↪  self._tex_path(texpath)) + "\n\\makeatother"
        return (g1, ..., ...)
```

Listing 7: TeXEmitter writes sanitized macro names and caches (python/lpm_paths/emitters/tex.py).

Every declared path yields:

- `lp@path@coords@<safe>` — the coordinate sequence consumed by TikZ.
- `lp@path@upmarklabels@<safe>` — pre-formatted `\node` commands.
- `lp@path@insidecornercoord@<safe>@<i>` — positional lookup for `\highlightInsideCorner`.
- `lp@path@gridsize@<safe>` — bounding box for `\drawGrid`.

### 5.5 Drawing layer

```
\newcommand\drawLatticePath[2][]{%
  \begingroup
    \lp@lpath@labelupmarksfalse
    \lp@lpath@showinsidecornersfalse
    \tikzset{#1}%
    \def\lp@readyflag{0}%
    \ifcsname lp@path@ready@#2\endcsname
      \edef\lp@readyflag{\csname lp@path@ready@#2\endcsname}%
    \fi
    \if\lp@readyflag1%
      \draw[lp/path,lp/lpath,#1] plot coordinates
        { \csname lp@path@coords@#2\endcsname };%
      \iflp@lpath@labelupmarks \csname lp@path@upmarklabels@#2\endcsname \fi
      \iflp@lpath@showinsidecorners \csname
      ↪  lp@path@insidecornerlabels@#2\endcsname \fi
      \endgroup
    \else
      \endgroup
      \lp@warn{Coordinates 'lp@path@coords@#2' not ready;}
    \fi
}
```

Listing 8: Fail-safe drawing routine (tex/latex/lpmres/lpmres-lpath.code.tex).

Shading and between-path outlines share nearly identical logic (`tex/latex/lpmres/lpmres-between.code.t`

```
\newcommand\shadeBetween[3][]{%
  \begingroup
    \def\lp@readyflag{0}%
    \ifcsname lp@between@ready@#2@#3\endcsname
      \edef\lp@readyflag{\csname lp@between@ready@#2@#3\endcsname}%
    \fi
    \if\lp@readyflag1%
      \endgroup
      \fill[#1] plot coordinates { \csname lp@between@coords@#2@#3\endcsname };%
    \else
      \endgroup
      \lp@warn{Between region (#2,#3) not ready;}
    \fi
}
```

Listing 9: Shading helper uses cached polygons (tex/latex/lpmres/lpmres-between.code.tex).

# 6 Macro reference

## 6.1 Declaring and referencing paths

- `\lpDeclarePath{name}{bits}` — declare or overwrite a path. Names are sanitized to `[A-Za-z0-9_]` and collisions trigger a package warning.
- `\drawLatticePath[options]{name}` — draw cached coordinates. Options propagate to TikZ. Available feature toggles:
  - `lplpath/label upmarks` — show step indices at upmarks
  - `lplpath/show inside corners` — highlight all inside corners in red
  - `lplpath/show endpoints` — mark start point (0,0) and endpoint (#zeros, #ones) in blue
- `\drawGrid[style]{name}` — mirror the bounding box of the declared path. Falls back to silently doing nothing on the first pass.
- `\highlightInsideCorner[style]{name}{index}` — pinpoint a single corner using the cached coordinate macros.

## 6.2 Schubert picture helpers

The light-weight environment defined in `tex/latex/lpmres/lpmres-pic.code.tex` avoids repetitive TikZ boilerplate:

```
\newenvironment{schubertpic}[1][]{%
  \begin{tikzpicture}[x=0.6cm,y=0.6cm, #1]%
  \draw[gray!30] (0,0) grid +(20,20);%
  \draw[->] (0,0)--+(21,0); \draw[->] (0,0)--+(0,21);%
}{%
  \end{tikzpicture}%
}
```

Listing 10: Default diagram scaffold (tex/latex/lpmres/lpmres-pic.code.tex).

Use nested `\begin{scope}` layers to mix multiple shaded regions in the same axes.

# 7 Python module details

## 7.1 Between-region construction

The polygon builder ensures walkable regions and removes duplicate points:

```python
def between_polygon(L_bits: str, U_bits: str) -> List[Coord]:
    L = makeLatticePath(L_bits)
    U = makeLatticePath(U_bits)
    if L.coords[-1] != U.coords[-1]:
        raise InputSpecError("Paths must share the same endpoint.")
    upper = U.coords[:]
    lower = list(reversed(L.coords))[1:-1]
    polygon = upper + lower + [upper[0]]
    dedup: List[Coord] = []
    for c in polygon:
        if not dedup or dedup[-1] != c:
            dedup.append(c)
    return dedup
```

Listing 11: `between_polygon` emits a closed polygon (python/lpm_paths/between.py).

## 7.2 Cache fencing

```python
@dataclass(frozen=True)
class Cache:
    root: str

    @staticmethod
    def make(root: Optional[str] = None) -> "Cache":
        r = root or DEFAULT_CACHE_DIR
        ensure_dir(r)
        return Cache(root=r)

    def guardPath(self, path: str) -> str:
        root_real = os.path.realpath(self.root)
        path_real = os.path.realpath(path)
        common = os.path.commonpath([root_real, path_real])
        if common != root_real:
            raise CacheFenceError(f"Path escapes fence: {path}")
        return path_real
```

Listing 12: Cache sandbox prevents `../../` escapes (python/lpm_paths/cache.py).

Every emitted file stays under `lp-cache/`, and the helper also provides `tex_path` for user-friendly relative paths inside warnings.

## 7.3 Manifest serialization and hashing

Emitted JSON manifests are deterministic because `lpm_paths.hashing.key_of` normalizes input via canonical JSON before running BLAKE2b, ensuring cache hits survive between platforms or when file names change.

# 8 Caching and housekeeping

The cache layout mirrors typical CTAN distributions: `lp-cache/path-*.tex`, `lp-cache/path-*.json`, `lp-cache/between-*.tex`, plus a `.names/` directory that tracks the unsanitized names responsible for each path identifier. When iterating on the Python module or collecting assets for publication, use `scripts/clean-cache.sh`:

```
#!/bin/bash
set -e
echo "Cleaning lpmresonance build artifacts and cache..."
find "$PROJECT_ROOT" -type f \( -name "*.aux" -o -name "*.log" \
  -o -name "*.pytxcode" -o -name "*.fdb_latexmk" \) -delete
find "$PROJECT_ROOT" -type d -name "pythontex-files-*" -exec rm -rf {} +
find "$PROJECT_ROOT" -type d -name "_minted-*" -exec rm -rf {} +
find "$PROJECT_ROOT" -type d -name "lp-cache" -exec rm -rf {} +
find "$PROJECT_ROOT" -type d -name ".names" -exec rm -rf {} +
```

Listing 13: Automated cleanup before packaging (scripts/clean-cache.sh).

# 9 Worked diagrams

## 9.1 Single annotated path

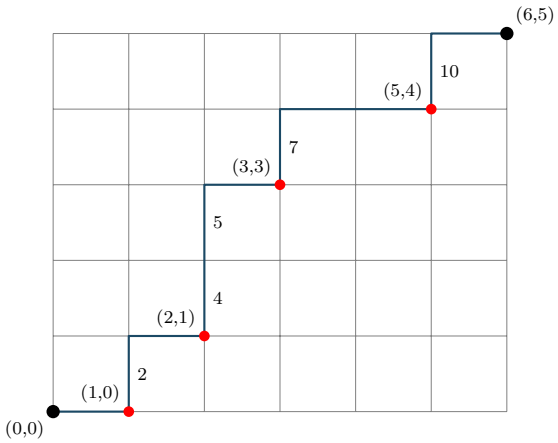


Figure 1: Named path with upmark labels, highlighted inside corners, and endpoints.

The figure stems from the following code fragment:

```latex
\lpDeclarePath{docDemo}{01011010010}
\begin{tikzpicture}
  \drawGrid{docDemo}
  \drawLatticePath[
    lplpath/label upmarks,
    lplpath/show inside corners,
    lplpath/show endpoints,
    lp/lpath/.style = {LPBlue, thick},
  ]{docDemo}
\end{tikzpicture}
```

Listing 14: Code used to generate Figure 1.

### 9.2 Between-region shading

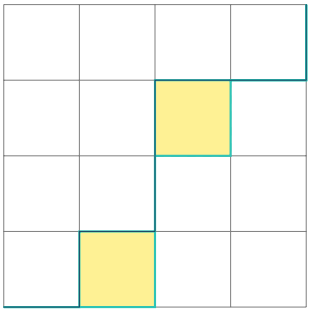

Figure 2: Region sandwiched between two bit strings.

The supporting code mirrors the declarative style encouraged throughout the package:

```latex
\shadeBetweenBits{00110101}{01011001}{DocLower}{DocUpper}
\begin{tikzpicture}
  \begin{scope}[on background layer]
    \shadeBetween[LPHighlight!60]{DocLower}{DocUpper}
  \end{scope}
  \drawBetween[LPTeal, thick]{DocLower}{DocUpper}
  \drawLatticePath[lp/lpath/.style={LPTeal,dashed}]{docLower}
  \drawLatticePath[lp/lpath/.style={LPBlue}]{docUpper}
\end{tikzpicture}
```

Listing 15: Code responsible for Figure 2.

## 10 Testing, troubleshooting, and release checklist

- **Unit tests:** `pytest tests/python` validates geometry invariants and manifest serialization. Extend this suite whenever Python helpers gain new fields.
- **End-to-end sanity:** compile `examples/example.tex` or `examples/test-features.tex` with `latexmk -pdf -shell-escape`. Confirm that `lp-cache/` gains fresh assets and no

warnings show up in the log.

- **Docs and changelog:** refresh `docs/` (the MkDocs site) plus this LaTeX file when user-facing behavior changes. Keep `CHANGELOG.md` synchronized with releases.
- **Packaging:** run `scripts/clean-cache.sh`, bump the version, tag the release, and upload to PyPI/CTAN.

## 11 Future refinements

1. Refine minted listings by splitting implementation details into an appendix once the document leaves draft status.
2. Automate screenshot/figure regeneration via a dedicated `latexmk` target so the diagrams double as regression tests.
3. Track cache versioning in `EMITTER_VERSION` and surface it inside the documentation to help downstream users diagnose stale assets.

**Acknowledgments.** The repository already hosts an extensive MkDocs site; this CTAN-style document serves as a printable snapshot that mirrors those instructions while surfacing the underlying mechanism in greater detail.