PAL API Reference

Generated from public `Protocol` declarations and source docstrings. Each description is maintained beside its source signature. Regenerate with `python Documentation/generators/generate_api_reference.py`; use `--check` to detect drift. Consult the PAL Guide for worked usage and Architecture and Design for implementation rationale.

Axis shorthand: `...` after an X/x-specific name includes its Y/y and Z/z variants. Each variant remains a separate callable or property.

Factories and utilities

Function	Description
`FastMode()`	Select PAL fast mode before constructing any PAL object or window; the process-wide mode cannot be changed after construction.
`Seed(seed)`	Seed PAL's shared random-number stream; repeatability also depends on the sequence of random calls.
`Random()`	Draw a uniform random value from PAL's shared random-number stream.
`RandInt(max)`	Draw an integer from PAL's random stream in the supported range below the positive bound `max`.
`AwaitWindows(timeout=30)`	Wait for PAL windows that are still initializing; raises if a window exits early or initialization times out.
`ColorScale(colors, value)`	Map a scalar value to an RGB color by interpolating through the supplied color scale.
`NewIList()`	Create an empty PAL integer list usable from Python and supported compiled functions.
`MooreHood(dim, excludeCenter=False)`	Build relative integer offsets for a Moore neighborhood in `dim` dimensions; optionally exclude the center offset.
`VonNeumannHood(dim, excludeCenter=False)`	Build relative integer offsets for a Von Neumann neighborhood in `dim` dimensions; optionally exclude the center offset.
`CircleHood(dim, rad, excludeCenter=False)`	Build relative integer offsets within radius `rad` in `dim` dimensions; optionally exclude the center offset.
`NewMultinomial(other=None)`	Create a PAL multinomial/binomial sampler, optionally reusing configuration from `other`.
`NewGrid(dimensions, dtype)`	Create a typed lattice grid with the requested dimensions and element dtype.
`NewAgentGrid(dimensions, numAgentProps=0, isStackable=False)`	Create a grid for individual agents; `numAgentProps` sets the number of per-agent properties and `isStackable` controls whether multiple agents may occupy a site.

Function	Description
`NewPopGrid(dimensions, dtype=np.int64)`	Create a lattice of integer population counts; `dtype` explicitly selects `uint8`, `uint16`, `uint32`, or `int64` (default) field storage, with `int16`, `int32`, `int64`, or `int64` deltas respectively.
`NewPDEgrid(dimensions)`	Create a lattice-backed scalar field for numerical transport operations.

Multinomial

Kind	Declaration	Description
method	`Binomial(self, n: int, p: float) -> int`	Draw a binomial sample with n trials and success probability p using this sampler's configured random stream.
method	`Setup(self, n: int)`	Configure the sampler for `n` trials before repeated sampling.
method	`Sample(self, p: float) -> int`	Draw a binomial sample using the sampler's current configuration and success probability p.

IList

Kind	Declaration	Description
method	`__len__(self) -> int`	Return the number of integers currently stored in the list.
method	`__getitem__(self, index: int) -> int`	Return the integer stored at the given list position.
method	`Append(self, i: int)`	Append integer `i` to the end of the list.
method	`Clear(self)`	Remove all values from the list.
method	`Random(self) -> int`	Return a randomly selected value from the list; the list must be nonempty.
method	`Shuffle(self)`	Shuffle the list in place using PAL's random stream.
method	`All(self) -> np.ndarray`	Return a detached NumPy array containing the list values.
method	`Iter(self) -> np.ndarray`	Iterate over the current list values without creating the detached copy returned by All().

AgentGrid

Kind	Declaration	Description
method	`__len__(self) -> int`	Return the number of living agents in the grid.

Kind	Declaration	Description
property	`xDim...(self) -> int`	Number of lattice sites along x; corresponding Y/Z variants use the same contract.
property	`nDims(self) -> int`	Number of spatial dimensions.
property	`wrapX...(self) -> bool`	Whether x uses periodic boundary wrapping; corresponding Y/Z variants use the same contract.
method	`Box(self, x1: int, x2: int, y1: int=None, y2: int=None, z1: int=None, z2: int=None)`	Iterate lattice coordinates in the half-open rectangular region [x1, x2) × [y1, y2) × [z1, z2); omit unused axes in lower dimensions.
method	`Hood(self, hood, x: int, y: int=None, z: int=None, *, unroll: bool=False)`	Apply relative neighborhood offsets at a lattice coordinate, respecting wrapped and nonwrapped boundaries. `unroll=True` enables compiled-loop expansion where supported.
method	`GetPop(self) -> int`	Return the number of living agents in the grid.
method	`Alive(self, agent: int) -> bool`	Return whether agent currently identifies a living agent.
method	`NewAgentSQ(self, x: int, y: int=-1, z: int=-1) -> int`	Create an agent at the center of a lattice site specified by integer coordinates or a supported linear site index.
method	`NewAgent(self, x: float, y: float=-1.0, z: float=-1.0) -> int`	Create an agent at the supplied continuous spatial coordinates.
method	`Dispose(self, agent: int)`	Remove a living agent from the grid; its handle must not be used as a live agent afterward.
method	`I(self, agent: int) -> int`	Return the linear lattice-site index containing the agent.
method	`XSQ...(self, agent: int) -> int`	Return the agent's integer x site coordinate; corresponding Y/Z variants use the same contract.
method	`X...(self, agent: int) -> float`	Return the agent's continuous x coordinate; corresponding Y/Z variants use the same contract.
method	`__getitem__(self, key) -> float`	Read a supported per-agent property value.
method	`__setitem__(self, key, value: float)`	Set a supported per-agent property value for the selected agent/property key.
method	`ToI(self, x: int, y: int=-1, z: int=-1) -> int`	Convert lattice coordinates to the corresponding linear site index.
method	`ItoX...(self, i: int) -> int`	Convert a linear site index to its x coordinate; corresponding Y/Z variants use the same contract.
method	`InWrapSQX...(self, value: int) -> int`	Wrap an integer x site coordinate when x is periodic; corresponding Y/Z variants use the same contract.
method	`InWrapX...(self, value: float) -> float`	Wrap a continuous x coordinate when x is periodic; corresponding Y/Z variants use the same contract.

Kind	Declaration	Description
method	`DispWrapX(self, x1: float, x2: float) -> float`	Return the shortest signed x displacement under periodic wrapping.
method	`DispWrapY(self, y1: float, y2: float) -> float`	Return the shortest signed y displacement under periodic wrapping.
method	`DispWrapZ(self, z1: float, z2: float) -> float`	Return the shortest signed z displacement under periodic wrapping.
property	`counts(self) -> object`	Read-only occupancy counts for lattice sites; change occupancy through agent creation, movement, and disposal methods.
method	`LastAgent(self, x: int, y: int=-1, z: int=-1) -> int`	Return the last agent handle at the specified lattice site, or the empty-site sentinel when none is present.
method	`MoveSQ(self, agent: int, x: int, y: int=-1, z: int=-1)`	Move an agent to a lattice site specified by integer coordinates or a supported linear site index.
method	`Move(self, agent: int, x: float, y: float=-1.0, z: float=-1.0)`	Move an agent to continuous coordinates; use `MoveSQ` when movement should target a lattice site.
method	`AgentsAt(self, x: int, y: int=-1, z: int=-1)`	Return the agent handle or handles occupying the specified lattice site.
method	`AgentsInRadius(self, rad: float, x: float, y: float=-1.0, z: float=-1.0, *, exclude: int=None)`	Return agents within radius `rad` of a continuous position; `exclude` can omit one agent from the results.
method	`All(self, shuffle: bool=False) -> np.ndarray`	Return a snapshot of living agent handles, optionally shuffled. The snapshot fixes membership, not later world state or occupancy queries.

Grid

Kind	Declaration	Description
method	`__len__(self) -> int`	Return the number of lattice sites.
method	`__getitem__(self, key)`	Read the value or state selected by the supported index or coordinate key.
method	`__setitem__(self, key, value)`	Assign the value or state selected by the supported index or coordinate key.
property	`xDim...(self) -> int`	Number of lattice sites along x; corresponding Y/Z variants use the same contract.
property	`nDims(self) -> int`	Number of spatial dimensions.

Kind	Declaration	Description
property	`wrapX...(self) -> bool`	Whether x uses periodic boundary wrapping; corresponding Y/Z variants use the same contract.
method	`Box(self, x1: int, x2: int, y1: int=None, y2: int=None, z1: int=None, z2: int=None)`	Iterate lattice coordinates in the half-open rectangular region [x1, x2) × [y1, y2) × [z1, z2); omit unused axes in lower dimensions.
method	`Hood(self, hood, x: int, y: int=None, z: int=None, *, unroll: bool=False)`	Apply relative neighborhood offsets at a lattice coordinate, respecting wrapped and nonwrapped boundaries. `unroll=True` enables compiled-loop expansion where supported.
method	`ToI(self, x: int, y: int=-1, z: int=-1) -> int`	Convert lattice coordinates to a linear site index.
method	`ItoX...(self, i: int) -> int`	Convert a linear site index to its x coordinate; corresponding Y/Z variants use the same contract.
method	`InWrapX...(self, value: int) -> int`	Wrap an x coordinate when x is periodic; otherwise return it unchanged; corresponding Y/Z variants use the same contract.

PopGrid

Kind	Declaration	Description
method	`__len__(self) -> int`	Return the number of lattice sites.
property	`xDim...(self) -> int`	Number of lattice sites along x; corresponding Y/Z variants use the same contract.
property	`nDims(self) -> int`	Number of spatial dimensions.
property	`wrapX...(self) -> bool`	Whether x uses periodic boundary wrapping; corresponding Y/Z variants use the same contract.
method	`Box(self, x1: int, x2: int, y1: int=None, y2: int=None, z1: int=None, z2: int=None)`	Iterate lattice coordinates in the half-open rectangular region [x1, x2) × [y1, y2) × [z1, z2); omit unused axes in lower dimensions.
method	`Hood(self, hood, x: int, y: int=None, z: int=None, *, unroll: bool=False)`	Apply relative neighborhood offsets at a lattice coordinate, respecting wrapped and nonwrapped boundaries. `unroll=True` enables compiled-loop expansion where supported.
method	`ToI(self, x: int, y: int=-1, z: int=-1) -> int`	Convert lattice coordinates to a linear site index.
method	`ItoX...(self, i: int) -> int`	Convert a linear site index to its x coordinate; corresponding Y/Z variants use the same contract.

Kind	Declaration	Description
method	`InWrapX...(self, value: int) -> int`	Wrap an x coordinate when x is periodic; otherwise return it unchanged; corresponding Y/Z variants use the same contract.
method	`__getitem__(self, key) -> int`	Return the integer population count at the selected site.
method	`__setitem__(self, key, value: int)`	Set the population count at the selected site immediately; this is distinct from queueing a delta with `Add()`.
method	`Add(self, value: int, x: int, y: int=-1, z: int=-1)`	Queue a signed population-count delta at a site; call `Update()` to apply pending deltas.
method	`Update(self)`	Apply queued population-count deltas to the current counts.
method	`Reset(self)`	Clear population counts and any pending deltas.
method	`GetPop(self) -> int`	Return the total population summed across all lattice sites.
method	`All(self) -> np.ndarray`	Return linear indices of sites whose population count is nonzero.

PDEgrid

Kind	Declaration	Description
method	`__len__(self) -> int`	Return the number of lattice sites.
property	`xDim...(self) -> int`	Number of lattice sites along x; corresponding Y/Z variants use the same contract.
property	`nDims(self) -> int`	Number of spatial dimensions.
property	`wrapX...(self) -> bool`	Whether x uses periodic boundary wrapping; corresponding Y/Z variants use the same contract.
method	`Box(self, x1: int, x2: int, y1: int=None, y2: int=None, z1: int=None, z2: int=None)`	Iterate lattice coordinates in the half-open rectangular region [x1, x2) × [y1, y2) × [z1, z2); omit unused axes in lower dimensions.
method	`Hood(self, hood, x: int, y: int=None, z: int=None, *, unroll: bool=False)`	Apply relative neighborhood offsets at a lattice coordinate, respecting wrapped and nonwrapped boundaries. `unroll=True` enables compiled-loop expansion where supported.
method	`ToI(self, x: int, y: int=-1, z: int=-1) -> int`	Convert lattice coordinates to a linear site index.
method	`ItoX...(self, i: int) -> int`	Convert a linear site index to its x coordinate; corresponding Y/Z variants use

Kind	Declaration	Description
		the same contract.
method	`InWrapX...(self, value: int) -> int`	Wrap an x coordinate when x is periodic; otherwise return it unchanged; corresponding Y/Z variants use the same contract.
method	`__getitem__(self, key) -> float`	Return the field value at the selected site.
method	`__setitem__(self, key, value: float)`	Set the field value at the selected site immediately; this is distinct from queueing a change with `Add()`.
method	`Add(self, value: float, x: int, y: int=-1, z: int=-1)`	Queue a field-value delta at a site; call `Update()` to apply pending changes.
method	`Update(self)`	Apply queued field changes to the current field.
method	`Reset(self)`	Clear the field and any pending changes.
method	`SetTimeSpaceStep(self, dt: float, dx: float, dy: float=1.0, dz: float=1.0)`	Set the temporal step `dt` and spatial steps `dx`, `dy`, and `dz` used by numerical transport methods.
method	`Dx...(self) -> float`	Return the configured spatial step along x; corresponding Y/Z variants use the same contract.
method	`Dt(self) -> float`	Return the configured time step.
method	`Diffusion(self, rateConstant: float, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply scalar-coefficient diffusion using the configured grid spacing and optional per-face boundary conditions.
method	`DiffusionMask(self, rateConstant: float, mask=None, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Diffuse across unmasked interfaces, with optional boundary values on active domain faces; masked cells remain impermeable.
method	`DiffusionField(self, rateConstants, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply diffusion with spatially varying rate constants and optional boundary conditions.
method	`DiffusionInterfaces(self, rateConstantsX, rateConstantsY=None, rateConstantsZ=None, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply diffusion using rate constants defined on grid interfaces; separate arrays may specify x, y, and z interfaces.
method	`DiffusionADI(self, rateConstant: float, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply diffusion using an alternating-direction implicit method.

Kind	Declaration	Description
method	`DiffusionRadialCircle(self, rateConstant: float, outerBC=None)`	Apply radial diffusion on a circular domain with an optional outer boundary condition.
method	`DiffusionRadialSphere(self, rateConstant: float, outerBC=None)`	Apply radial diffusion on a spherical domain with an optional outer boundary condition.
method	`Advection(self, vx: float, vy: float=0.0, vz: float=0.0, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply advection using a constant velocity vector and optional per-face boundary conditions.
method	`AdvectionField(self, xVels, yVels=None, zVels=None, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply advection using spatially varying velocity fields.
method	`AdvectionInterfaces(self, xVels, yVels=None, zVels=None, xMinBC=None, xMaxBC=None, yMinBC=None, yMaxBC=None, zMinBC=None, zMaxBC=None)`	Apply advection using velocity values defined on grid interfaces.

PixWindow visualization

```
StartPixWindow(xDim, yDim, scale=1, title='PAL', headless=False) -> tuple[Pix, PixWindow]
```

Create a pixel-based visualization window and its writable pixel buffer.

Kind	Declaration	Description
method	`IsOpen(self) -> bool`	Return whether the window is currently open.
method	`Update(self)`	Publish the current pixel buffer as a frame.
method	`Save(self, path: str, block=False)`	Save the current pixel buffer to an image file; block controls whether to wait for completion.
method	`StartGif(self, path: str, delay=100)`	Start recording frames to a GIF at path with the requested frame delay in milliseconds.
method	`AddGifFrame(self, block=False)`	Append the current pixel buffer as a GIF frame; optionally wait for the frame operation to finish.
method	`StopGif(self)`	Finish GIF recording and write the accumulated animation.
method	`Close(self)`	Close the window and release its rendering resources.

OpenGLWindow visualization

```
StartOpenGLWindow(xDim, yDim, zDim=None, width=800, height=800, title='PAL', headless=False) ->
```

```
tuple[OpenGLDraw, OpenGLWindow]
```

Create an OpenGL visualization window for 2D or 3D drawing.

Kind	Declaration	Description
method	`Update(self)`	Publish the accumulated drawing primitives as the next frame.
method	`IsOpen(self) -> bool`	Return whether the window is currently open.
method	`StartGif(self, path, delay=100)`	Start recording rendered frames to a GIF with the requested frame delay in milliseconds.
method	`AddGifFrame(self, block=False, timeout=30)`	Capture the current scene as a GIF frame; optionally wait for completion.
method	`StopGif(self, timeout=30)`	Finish GIF recording and write the animation.
method	`Borders(self, width, color)`	Draw persistent outline borders around the model domain.
method	`Camera(self, x, y, z, yaw=None, pitch=None)`	Set the 3D camera position and optional yaw/pitch angles.
method	`Clear(self)`	Clear accumulated drawing primitives for the next frame.
method	`Background(self, color)`	Set the background color used when rendering frames.
method	`Save(self, path: str, block=False)`	Save the currently published scene to an image file; block controls whether to wait for completion.
method	`Close(self)`	Close the window and release rendering resources.

The `OpenGLWindow` table follows its source Protocol, which also contains method implementations; repeated declarations are listed once. `OpenGLDraw` is a compiled primitive collector, while the returned window handles rendering and output. Consult the PAL Guide for which object to use in Python and compiled drawing code.

Pix drawing buffer

Kind	Declaration	Description
method	`__setitem__(self, key, color)`	Write a packed RGB color to a pixel by (x, y) coordinates or a linear pixel index.
method	`xDim(self)`	Return the pixel buffer width.
method	`yDim(self)`	Return the pixel buffer height.
method	`ToI(self, x, y)`	Convert pixel coordinates (x, y) to a linear index.
method	`ItoX(self, i)`	Convert a linear pixel index to its x coordinate.
method	`ItoY(self, i)`	Convert a linear pixel index to its y coordinate.

Kind	Declaration	Description
method	`__len__(self)`	Return the total number of pixels.

OpenGLDraw primitive collector

Kind	Declaration	Description
method	`Clear(self)`	Discard all primitives accumulated for the next frame.
method	`Circle(self, rad, color, x, y, z=0.0)`	Add a circle centered at (x, y, z) with radius rad and color.
method	`Box(self, xLen, color, x, y, z=0.0, yLen=-1.0, zLen=-1.0)`	Add a box centered at (x, y, z); negative side lengths default to xLen.
method	`BoxSQ(self, color, x, y, z=0.0)`	Add a unit box centered on lattice site (x, y, z).
method	`Line(self, width, color, x1, y1, x2, y2, z1=0.0, z2=0.0)`	Add a line segment between endpoints with the requested width and color.

Scope and validation

This reference lists public Protocol signatures, not internal safe/fast jitclass methods. A signature does not capture every runtime overload, indexing form, or validation rule. See the PAL Guide and executable tests. Visualization constructors and window Protocol methods are included; Pix buffer and OpenGLDraw method signatures are also included; consult the PAL Guide for behavior.