API Reference

Contents

Simulation

Simulation

class tidy3d.Simulation

Contains all information about Tidy3d simulation.

Parameters:

- **center** (*Tuple[float, float, float] = (0.0, 0.0, 0.0)*) – [units = um]. Center of object in x, y, and z.
- **size** (*Tuple[Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf], Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf], Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf]] = None*) – [units = um]. Size in x, y, and z directions.
- **grid_size** (*Tuple[Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]]] = None*) – [units = um]. If

components are float, uniform grid size along x, y, and z. If components are array like, defines an array of nonuniform grid sizes centered at the simulation center . Note: if supplied sizes do not cover the simulation size, the first and last sizes are repeated to cover size.

- **medium** (*Union[tidy3d.components.medium.Medium, tidy3d.components.medium.AnisotropicMedium, tidy3d.components.medium.PECMedium, tidy3d.components.medium.PoleResidue, tidy3d.components.medium.Sellmeier, tidy3d.components.medium.Lorentz, tidy3d.components.medium.Debye, tidy3d.components.medium.Drude] = Medium(name=None, frequency_range=None, type='Medium', permittivity=1.0, conductivity=0.0)*) – Background medium of simulation, defaults to vacuum if not specified.
- **run_time** (*NonNegativeFloat = 0.0*) – [units = sec]. Total electromagnetic evolution time in seconds. Note: If simulation 'shutoff' is specified, simulation will terminate early when shutoff condition met.
- **structures** (*List[Structure] = []*) – List of structures present in simulation. Note: Structures defined later in this list override the simulation material properties in regions of spatial overlap.
- **sources** (*List[Union[tidy3d.components.source.VolumeSource, tidy3d.components.source.PlaneWave, tidy3d.components.source.ModeSource, tidy3d.components.source.GaussianBeam]] = []*) – List of electric current sources injecting fields into the simulation.
- **monitors** (*List[Union[tidy3d.components.monitor.FieldMonitor, tidy3d.components.monitor.FieldTimeMonitor, tidy3d.components.monitor.FluxMonitor, tidy3d.components.monitor.FluxTimeMonitor, tidy3d.components.monitor.ModeMonitor]] = []*) – List of monitors in the simulation. Note: monitor names are used to access data after simulation is run.
- **pml_layers** (*Tuple[Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber, NoneType], Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber, NoneType], Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber, NoneType]] = (None, None, None)*) – Specifications for the absorbing layers on x, y, and z edges. If `None`, no absorber will be added on that dimension and periodic boundary conditions will be used.
- **symmetry** (*Tuple[typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1]] = (0, 0, 0)*) – Tuple of integers defining reflection symmetry across a plane bisecting the simulation domain normal to the x-, y-, and z-axis, respectvely. Each element can be `0` (no symmetry), `1` (even, i.e. 'PMC' symmetry) or `-1` (odd, i.e. 'PEC' symmetry). Note that the vectorial nature of the fields must be taken into account to correctly determine the symmetry value.
- **shutoff** (*NonNegativeFloat = 1e-05*) – Ratio of the instantaneous integrated E-field intensity to the maximum value at which the simulation will automatically terminate time stepping. Used to prevent extraneous run time of simulations with fully decayed fields. Set to `0` to disable this feature.
- **subpixel** (*bool = True*) – If `True`, uses subpixel averaging of the permittivity based on structure definition, resulting in much higher accuracy for a given grid size.
- **courant** (*ConstrainedFloatValue = 0.9*) – Courant stability factor, controls time step to spatial step ratio. Lower values lead to more stable simulations for dispersive materials, but result in longer simulation times.

Example

```
>>> sim = Simulation(
...     size=(2.0, 2.0, 2.0),
...     grid_size=(0.1, 0.1, 0.1),
...     run_time=40e-11,
...     structures=[
...             Structure(
...                     geometry=Box(size=(1, 1, 1), center=(-1, 0, 0)),
...                     medium=Medium(permittivity=2.0),
...             ),
...     ],
...     sources=[
...             VolumeSource(
...                     size=(0, 0, 0),
...                     center=(0, 0.5, 0),
...                     polarization="Hx",
...                     source_time=GaussianPulse(
...                             freq0=2e14,
...                             fwidth=4e13,
...                     ),
...             )
...     ],
...     monitors=[
...             FieldMonitor(size=(0, 0, 0), center=(0, 0, 0), freqs=[1.5e14, 2e14], name='point'),
...             FluxMonitor(size=(1, 1, 0), center=(0, 0, 0), freqs=[2e14, 2.5e14], name='flux'),
...     ],
...     symmetry=(0, 0, 0),
...     pml_layers=(
...             PML(num_layers=20),
...             PML(num_layers=30),
...             None,
...     ),
...     shutoff=1e-6,
...     courant=0.8,
...     subpixel=False,
... )
```

▶ Show JSON schema

Fields:

- `center (Tuple[float, float, float])`
- `courant (float)`
- `grid_size (Tuple[Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]]])`
- `medium (Union[tidy3d.components.medium.Medium, tidy3d.components.medium.AnisotropicMedium, tidy3d.components.medium.PECMedium, tidy3d.components.medium.PoleResidue, tidy3d.components.medium.Sellmeier, tidy3d.components.medium.Lorentz, tidy3d.components.medium.Debye, tidy3d.components.medium.Drude])`
- `monitors (List[Union[tidy3d.components.monitor.FieldMonitor, tidy3d.components.monitor.FieldTimeMonitor, tidy3d.components.monitor.FluxMonitor, tidy3d.components.monitor.FluxTimeMonitor, tidy3d.components.monitor.ModeMonitor]])`
- `pml_layers (Tuple[Optional[Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber]], Optional[Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber]],`

`Optional[Union[tidy3d.components.pml.PML, tidy3d.components.pml.StablePML, tidy3d.components.pml.Absorber]]])`

- `run_time (pydantic.types.NonNegativeFloat)`
- `shutoff (pydantic.types.NonNegativeFloat)`
- `size (Tuple[Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf], Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf], Union[pydantic.types.NonNegativeFloat, tidy3d.components.types.Inf]])`
- `sources (List[Union[tidy3d.components.source.VolumeSource, tidy3d.components.source.PlaneWave, tidy3d.components.source.ModeSource, tidy3d.components.source.GaussianBeam]])`
- `structures (List[tidy3d.components.structure.Structure])`
- `subpixel (bool)`
- `symmetry (Tuple[typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1]])`

attribute* `center`*: Coordinate = (0.0, 0.0, 0.0)

Center of object in x, y, and z.

Constraints:
- **units** = um

attribute* `courant`*: float = 0.9

Validating setup

Courant stability factor, controls time step to spatial step ratio. Lower values lead to more stable simulations for dispersive materials, but result in longer simulation times.

Constraints:
- **exclusiveMinimum** = 0.0
- **maximum** = 1.0

attribute* `grid_size`*: Tuple[Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]], Union[pydantic.types.PositiveFloat, List[pydantic.types.PositiveFloat]]] [Required]

If components are float, uniform grid size along x, y, and z. If components are array like, defines an array of nonuniform grid sizes centered at the simulation center . Note: if supplied sizes do not cover the simulation size, the first and last sizes are repeated to cover size.

Constraints:
- **units** = um

attribute* `medium`*: Union[tidy3d.components.medium.Medium, tidy3d.components.medium.AnisotropicMedium, tidy3d.components.medium.PECMedium, tidy3d.components.medium.PoleResidue, tidy3d.components.medium.Sellmeier, tidy3d.components.medium.Lorentz, tidy3d.components.medium.Debye, tidy3d.components.medium.Drude] = Medium(name=None, frequency_range=None, type='Medium', permittivity=1.0, conductivity=0.0)

Background medium of simulation, defaults to vacuum if not specified.

***attribute* monitors*: List[Union[[tidy3d.components.monitor.FieldMonitor](), [tidy3d.components.monitor.FieldTimeMonitor](), [tidy3d.components.monitor.FluxMonitor](), [tidy3d.components.monitor.FluxTimeMonitor](), [tidy3d.components.monitor.ModeMonitor]()]] = []*

List of monitors in the simulation. Note: monitor names are used to access data after simulation is run.

Validated by:
- `objects_in_sim_bounds`
- `field_has_unique_names`

***attribute* pml_layers*: Tuple[Optional[Union[[tidy3d.components.pml.PML](), [tidy3d.components.pml.StablePML](), [tidy3d.components.pml.Absorber]()]], Optional[Union[[tidy3d.components.pml.PML](), [tidy3d.components.pml.StablePML](), [tidy3d.components.pml.Absorber]()]], Optional[Union[[tidy3d.components.pml.PML](), [tidy3d.components.pml.StablePML](), [tidy3d.components.pml.Absorber]()]]] = (None, None, None)*

Specifications for the absorbing layers on x, y, and z edges. If `None`, no absorber will be added on that dimension and periodic boundary conditions will be used.

Validated by:
- `_structures_not_close_pml`
- `set_none_to_zero_layers`

***attribute* run_time*: pydantic.types.NonNegativeFloat = 0.0*

Total electromagnetic evolution time in seconds. Note: If simulation 'shutoff' is specified, simulation will terminate early when shutoff condition met.

Constraints:
- **units** = sec
- **minimum** = 0

***attribute* shutoff*: pydantic.types.NonNegativeFloat = 1e-05*

Ratio of the instantaneous integrated E-field intensity to the maximum value at which the simulation will automatically terminate time stepping. Used to prevent extraneous run time of simulations with fully decayed fields. Set to `0` to disable this feature.

Constraints:
- **minimum** = 0

***attribute* size*: Size [Required]*

Size in x, y, and z directions.

Constraints:
- **units** = um

***attribute* sources*: List[Union[[tidy3d.components.source.VolumeSource](), [tidy3d.components.source.PlaneWave](), [tidy3d.components.source.ModeSource](), [tidy3d.components.source.GaussianBeam]()]] = []*

List of electric current sources injecting fields into the simulation.

Validated by:
- `_warn_sources_mediums_frequency_range`

- `_warn_grid_size_too_small`
- `objects_in_sim_bounds`
- `_plane_wave_homogeneous`
- `field_has_unique_names`

attribute **`structures`***: List[tidy3d.components.structure.Structure] = []*

List of structures present in simulation. Note: Structures defined later in this list override the simulation material properties in regions of spatial overlap.

Validated by:
- `_structures_not_at_edges`
- `_validate_num_mediums`
- `objects_in_sim_bounds`
- `field_has_unique_names`

attribute **`subpixel`***: bool = True*

If `True`, uses subpixel averaging of the permittivity based on structure definition, resulting in much higher accuracy for a given grid size.

attribute **`symmetry`***: Tuple[typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1], typing_extensions.Literal[0, -1, 1]] = (0, 0, 0)*

Tuple of integers defining reflection symmetry across a plane bisecting the simulation domain normal to the x-, y-, and z-axis, respectvely. Each element can be `0` (no symmetry), `1` (even, i.e. 'PMC' symmetry) or `-1` (odd, i.e. 'PEC' symmetry). Note that the vectorial nature of the fields must be taken into account to correctly determine the symmetry value.

`add_ax_labels_lims`(*axis: typing_extensions.Literal[0, 1, 2], ax: matplotlib.axes._axes.Axes, buffer: float = 0.3*)→ matplotlib.axes._axes.Axes

Sets the x,y labels based on **`axis`** and the extends based on **`self.bounds`**.

Parameters:
- **axis** (*int*) – Integer index into 'xyz' (0,1,2).
- **ax** (*matplotlib.axes._subplots.Axes*) – Matplotlib axes to add labels and limits on.
- **buffer** (*float = 0.3*) – Amount of space to place around the limits on the + and - sides.

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

`discretize`(*box: tidy3d.components.geometry.Box*)→ tidy3d.components.grid.Grid

Grid containing only cells that intersect with a **`Box`**.

Parameters: **box** (**`Box`**) – Rectangular geometry within simulation to discretize.

Returns: The FDTD subgrid containing simulation points that intersect with **`box`**.

`Grid`

Return type:

discretize_inds(*box: tidy3d.components.geometry.Box*) → List[Tuple[int, int]]

Start and stopping indexes for the cells that intersect with a `Box`.

Parameters: **box** (`Box`) – Rectangular geometry within simulation to discretize.

Returns: The (start, stop) indexes of the cells that intersect with `box` in each of the three dimensions.

Return type: List[Tuple[int, int]]

epsilon(*box: tidy3d.components.geometry.Box, coord_key: str = 'centers', freq: Optional[float] = None*) → Dict[str, xarray.core.dataarray.DataArray]

Get array of permittivity at volume specified by box and freq

Parameters:

- **box** (`Box`) – Rectangular geometry specifying where to measure the permittivity.
- **coord_key** (*str = 'centers'*) – Specifies at what part of the grid to return the permittivity at. Accepted values are `{'centers', 'boundaries', 'Ex', 'Ey', 'Ez'}`. The field values (eg. 'Ex') correspond to the correponding field locations on the yee lattice. If field values are selected, the corresponding epsilon component from the main diagonal of the epsilon tensor is returned. Otherwise, the average of the diagonal values is returned.
- **freq** (*float = None*) – The frequency to evaluate the mediums at. If not specified, evaluates at infinite frequency.

Returns: Mapping of coordinate type to xarray DataArray containing permittivity data. keys of dict are `{'centers', 'boundaries', 'Ex', 'Ey', 'Ez', 'Hx', 'Hy', 'Hz'}`. `'centers'` contains the permittivity at the yee cell centers. *'boundaries'*` contains the permittivity at the corner intersections between yee cells. `'Ex'` and other field keys contain the permittivity at the corresponding field position in the yee lattice. For details on xarray datasets, refer to [xarray's Documentaton](https://docs.xarray.dev/).

Return type: Dict[str, xarray.DataArray]

classmethod **from_bounds**(*rmin: Tuple[float, float, float], rmax: Tuple[float, float, float]*)

Constructs a `Box` from minimum and maximum coordinate bounds

Parameters:

- **rmin** (*Tuple[float, float, float]*) – (x, y, z) coordinate of the minimum values.
- **rmax** (*Tuple[float, float, float]*) – (x, y, z) coordinate of the maximum values.

Example

```
>>> b = Box.from_bounds(rmin=(-1, -2, -3), rmax=(3, 2, 1))
```

`get_monitor_by_name(name: str)→ tidy3d.components.monitor.Monitor`

Return monitor named 'name'.

`inside(x, y, z)→ bool`

Returns `True` if point `(x,y,z)` inside volume of geometry.

Parameters:
- **x** (*float*) – Position of point in x direction.
- **y** (*float*) – Position of point in y direction.
- **z** (*float*) – Position of point in z direction.

Returns: Whether point `(x,y,z)` is inside geometry.

Return type: bool

`intersections(x: Optional[float] = None, y: Optional[float] = None, z: Optional[float] = None)`

Returns shapely geometry at plane specified by one non None value of x,y,z.

Parameters:
- **x** (*float = None*) – Position of plane in x direction, only one of x,y,z can be specified to define plane.
- **y** (*float = None*) – Position of plane in y direction, only one of x,y,z can be specified to define plane.
- **z** (*float = None*) – Position of plane in z direction, only one of x,y,z can be specified to define plane.

Returns: List of 2D shapes that intersect plane. For more details refer to [Shapely's Documentaton](#).

Return type: List[shapely.geometry.base.BaseGeometry]

`intersects(other)→ bool`

Returns `True` if two `Geometry` have intersecting *.bounds*.

Parameters: **other** (`Geometry`) – Geometry to check intersection with.

Returns: Whether the rectangular bounding boxes of the two geometries intersect.

Return type: bool

`intersects_plane(x: Optional[float] = None, y: Optional[float] = None, z: Optional[float] = None)→ bool`

Whether self intersects plane specified by one non-None value of x,y,z.

Parameters:
- **x** (*float = None*) – Position of plane in x direction, only one of x,y,z can be specified to define plane.
- **y** (*float = None*) – Position of plane in y direction, only one of x,y,z can be specified to define plane.
- **z** (*float = None*) – Position of plane in z direction, only one of x,y,z can be specified to define plane.

Returns: Whether this geometry intersects the plane.

Return type: bool

`static parse_xyz_kwargs(**xyz)→ Tuple[typing_extensions.Literal[0, 1, 2], float]`

Turns x,y,z kwargs into index of the normal axis and position along that axis.

Parameters:
- **x** (*float = None*) – Position of plane in x direction, only one of x,y,z can be specified to define plane.
- **y** (*float = None*) – Position of plane in y direction, only one of x,y,z can be specified to define plane.
- **z** (*float = None*) – Position of plane in z direction, only one of x,y,z can be specified to define plane.

Returns: Index into xyz axis (0,1,2) and position along that axis.

Return type: int, float

`plot(x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs)→ matplotlib.axes._axes.Axes`

Plot geometry cross section at single (x,y,z) coordinate.

Parameters:
- **x** (*float = None*) – Position of plane in x direction, only one of x,y,z can be specified to define plane.
- **y** (*float = None*) – Position of plane in y direction, only one of x,y,z can be specified to define plane.
- **z** (*float = None*) – Position of plane in z direction, only one of x,y,z can be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****patch_kwargs** – Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to [Matplotlib's documentation]().

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

`plot_eps(x: float = None, y: float = None, z: float = None, freq: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs)→ matplotlib.axes._axes.Axes`

Plot each of simulation's components on a plane defined by one nonzero x,y,z coordinate. The permittivity is plotted in grayscale based on its value at the specified frequency.

Parameters:
- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.

- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **freq** (*float = None*) – Frequency to evaluate the relative permittivity of all mediums. If not specified, evaluates at infinite frequency.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to Matplotlib's documentation.

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

plot_grid(*x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None*) → matplotlib.axes._axes.Axes

Plot the cell boundaries as lines on a plane defined by one nonzero x,y,z coordinate.

Parameters:
- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

plot_monitors(*x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs*) → matplotlib.axes._axes.Axes

Plot each of simulation's monitors on a plane defined by one nonzero x,y,z coordinate.

Parameters:
- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.

- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to Matplotlib's documentation.

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

```
plot_pml(x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs)→ matplotlib.axes._axes.Axes
```

Plot each of simulation's absorbing boundaries on a plane defined by one nonzero x,y,z coordinate.

Parameters:

- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to Matplotlib's documentation.

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

```
plot_sources(x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs)→ matplotlib.axes._axes.Axes
```

Plot each of simulation's sources on a plane defined by one nonzero x,y,z coordinate.

Parameters:

- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.

- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to [Matplotlib's documentation](#).

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

plot_structures(*x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs*) → matplotlib.axes._axes.Axes

Plot each of simulation's structures on a plane defined by one nonzero x,y,z coordinate.

Parameters:

- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to [Matplotlib's documentation](#).

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

plot_structures_eps(*x: float = None, y: float = None, z: float = None, freq: float = None, cbar: bool = True, ax: matplotlib.axes._axes.Axes = None, **kwargs*) → matplotlib.axes._axes.Axes

Plot each of simulation's structures on a plane defined by one nonzero x,y,z coordinate. The permittivity is plotted in grayscale based on its value at the specified frequency.

Parameters:

- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.

- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **freq** (*float = None*) – Frequency to evaluate the relative permittivity of all mediums. If not specified, evaluates at infinite frequency.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to [Matplotlib's documentation](#).

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

plot_symmetries(*x: float = None, y: float = None, z: float = None, ax: matplotlib.axes._axes.Axes = None, **kwargs*)→ matplotlib.axes._axes.Axes

Plot each of simulation's symmetries on a plane defined by one nonzero x,y,z coordinate.

Parameters:

- **x** (*float = None*) – position of plane in x direction, only one of x, y, z must be specified to define plane.
- **y** (*float = None*) – position of plane in y direction, only one of x, y, z must be specified to define plane.
- **z** (*float = None*) – position of plane in z direction, only one of x, y, z must be specified to define plane.
- **ax** (*matplotlib.axes._subplots.Axes = None*) – Matplotlib axes to plot on, if not specified, one is created.
- ****kwargs** –

  Optional keyword arguments passed to the matplotlib patch plotting of structure. For details on accepted values, refer to [Matplotlib's documentation](#).

Returns: The supplied or created matplotlib axes.

Return type: matplotlib.axes._subplots.Axes

static **pop_axis**(*coord: Tuple[Any, Any, Any], axis: int*)→ Tuple[Any, Tuple[Any, Any]]

Separates coordinate at `axis` index from coordinates on the plane tangent to `axis`.

Parameters:
- **coord** (*Tuple[Any, Any, Any]*) – Tuple of three values in original coordinate system.
- **axis** (*int*) – Integer index into 'xyz' (0,1,2).

Returns: The input coordinates are separated into the one along the axis provided and the two on the planar coordinates, like `axis_coord, (planar_coord1, planar_coord2)`.

Return type: Any, Tuple[Any, Any]

static **unpop_axis**(*ax_coord: Any, plane_coords: Tuple[Any, Any], axis: int*) → Tuple[Any, Any, Any]

Combine coordinate along axis with coordinates on the plane tangent to the axis.

Parameters:
- **ax_coord** (*Any*) – Value along axis direction.
- **plane_coords** (*Tuple[Any, Any]*) – Values along ordered planar directions.
- **axis** (*int*) – Integer index into 'xyz' (0,1,2).

Returns: The three values in the xyz coordinate system.

Return type: Tuple[Any, Any, Any]

property **bounding_box**

Returns `Box` representation of the bounding box of a `Geometry`.

Returns: Geometric object representing bounding box.

Return type: `Box`

property **bounds**: *Tuple[Tuple[float, float, float], Tuple[float, float, float]]*

Returns bounding box min and max coordinates.

Returns: Min and max bounds packaged as `(minx, miny, minz), (maxx, maxy, maxz)`.

Return type: Tuple[float, float, float], Tuple[float, float float]

property **dt**: *float*

Simulation time step (distance).

Returns: Time step (seconds).

Return type: float

property **frequency_range**: *Tuple[Union[float, tidy3d.components.types.NegInf], Union[float, tidy3d.components.types.Inf]]*

Range of frequencies spanning all sources' frequency dependence.

Returns: Minumum and maximum frequencies of the power spectrum of the sources at 5 standard deviations.

Return type: Tuple[float, float]

property **geometry**

Box representation of self (used for subclasses of Box).

Returns: Instance of Box representing self's geometry.

Return type: Box

property **grid***: tidy3d.components.grid.Grid*

FDTD grid spatial locations and information.

Returns: Grid storing the spatial locations relevant to the simulation.

Return type: Grid

property **medium_map***: Dict[Union[tidy3d.components.medium.Medium, tidy3d.components.medium.AnisotropicMedium, tidy3d.components.medium.PECMedium, tidy3d.components.medium.PoleResidue, tidy3d.components.medium.Sellmeier, tidy3d.components.medium.Lorentz, tidy3d.components.medium.Debye, tidy3d.components.medium.Drude], pydantic.types.NonNegativeInt]*

Returns dict mapping medium to index in material. `medium_map[medium]` returns unique global index of `AbstractMedium` in simulation.

Returns: Mapping between distinct mediums to index in simulation.

Return type: Dict[`AbstractMedium`, int]

property **mediums***: Set[Union[tidy3d.components.medium.Medium, tidy3d.components.medium.AnisotropicMedium, tidy3d.components.medium.PECMedium, tidy3d.components.medium.PoleResidue, tidy3d.components.medium.Sellmeier, tidy3d.components.medium.Lorentz, tidy3d.components.medium.Debye, tidy3d.components.medium.Drude]]*

Returns set of distinct `AbstractMedium` in simulation.

Returns: Set of distinct mediums in the simulation.

Return type: Set[`AbstractMedium`]

property **num_pml_layers***: List[Tuple[float, float]]*

Number of absorbing layers in all three axes and directions (-, +).

Returns: List containing the number of absorber layers in - and + boundaries.

Return type: List[Tuple[float, float]]

property **pml_thicknesses***: List[Tuple[float, float]]*

Thicknesses (um) of absorbers in all three axes and directions (-, +)

Returns: List containing the absorber thickness (micron) in - and + boundaries.

Return type: List[Tuple[float, float]]

property **tmesh***: tidy3d.components.types.Array*