
Arborize

Release 4.0.0b6

Robin De Schepper

Mar 07, 2024

CONTENTS:

1	Getting Started	3
2	Model Definitions	7
3	Model Schematics	9
4	Model Builders	11
5	arborize	13
5.1	arborize package	13
6	Defining a model	23
6.1	Cable types	23
6.2	Synapse types	25
6.3	Drawing a schematic	26
	Python Module Index	27
	Index	29

Arborize lets you describe your model definitions, import schematics from multiple sources and build equivalent multicompartmental models on the supported backends (Arbor and NEURON).

GETTING STARTED

To build our first model, we start by creating a definition consisting of *soma* and *basal_dendrite*.

We'll begin by defining *hh* and *pas* in the soma, and *pas* in the dendrites:

```
from arborize import define_model

definition = define_model({
    "cable_types": {
        "soma": {
            "cable": {
                "Ra": 100,
                "cm": 1,
            },
            "mechanisms": {
                "hh": {
                    "gnabar": 0.12,
                    "gkbar": 0.036,
                    "gl": 0.0003,
                    "el": -54.3,
                },
            },
        },
        "basal_dendrite": {
            "cable": {
                "Ra": 100,
                "cm": 1,
            },
            "mechanisms": {
                "pas": {
                    "g": 0.001,
                    "e": -65,
                },
            },
        },
    },
})
```

Next up we need to get a schematic, download [this morphology](#) from NeuroMorpho as *morpho.swc*, then we can create a file schematic from it:

```
from arborize import file_schematic

schematic = file_schematic("morpho.swc", definition)
```

Hint: Arborize uses `MorphIO` to load schematics from file. The points are labelled with the `SectionType` enum.

We're ready to build a cell:

```
from arborize import neuron_build

cell = neuron_build(schematic)
```

Hint: Arborize's NEURON builder uses `Patch` to construct NEURON objects. It provides many convenience functions.

Let's record the soma and plot the results:

```
from patch import p
import plotly.express as px

r = cell.soma[0].record()
cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
t = p.time
p.celsius = 6.3
p.run(100)
px.line(x=list(r), y=list(t)).show()
```

To create variants of your cell model, pass the template definition to `define_model`:

```
variant = define_model(definition, {
    "cable_types": {
        "soma": {
            "mechanisms": {
                "hh": {
                    "gkbar": 0.172
                }
            }
        }
    }
})
```

We can now plot the difference between the 2 cell types:

```
from patch import p
import plotly.express as px

wildtype_schematic = file_schematic("morpho.swc", definition)
wildtype_cell = neuron_build(schematic)
variant_schematic = file_schematic("morpho.swc", variant)
variant_cell = neuron_build(variant_schematic)
```

(continues on next page)

(continued from previous page)

```
wildtype_cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
variant_cell.basal_dendrite[0].iclamp(delay=5, duration=1, amplitude=0.1, x=1)
r_wt = wildtype_cell.soma[0].record()
r_var = variant_cell.soma[0].record()
t = p.time
p.celsius = 6.3
p.run(100)
px.line(x=list(r), y=list(t)).show()
```


MODEL DEFINITIONS

MODEL SCHEMATICS

MODEL BUILDERS

ARBORIZE

5.1 arborize package

5.1.1 Subpackages

`arborize.builders` package

Module contents

`arborize.schematics` package

Module contents

5.1.2 Submodules

5.1.3 `arborize.definitions` module

```
class arborize.definitions.CableProperties(Ra: float = None, cm: float = None)
```

Bases: `Copy`, `Merge`, `Assert`, `Iterable`

Ra: `float = None`

cm: `float = None`

Axial resistivity in ohm/cm

```
class arborize.definitions.CablePropertiesDict
```

Bases: `dict`

Ra: `float`

cm: `float`

```
class arborize.definitions.CableType(cable_property_class=<class  
                                     'arborize.definitions.CableProperties'>)
```

Bases: `object`

add_ion(*key: str, ion: Ion*)

add_mech(*mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str], mech: Mechanism*)

add_synapse(*label: str | tuple[str] | tuple[str, str] | tuple[str, str, str], synapse: Synapse*)

```
classmethod anchor(defs: ~typing.Iterable[~arborize.definitions.CableType], synapses:
    dict[typing.Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
    arborize.definitions.Synapse] | None = None, use_defaults: bool = False,
    ion_class=<class 'arborize.definitions.Ion'>) → CableType

assert_()

cable: CableProperties

copy()

classmethod default(ion_class=<class 'arborize.definitions.Ion'>)

ions: dict[str, arborize.definitions.Ion]

mechs: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
    arborize.definitions.Mechanism]

merge(def_right: CableType)

set(param: Parameter)

synapses: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
    arborize.definitions.Synapse]

class arborize.definitions.CableTypeDict
    Bases: dict
    cable: CablePropertiesDict
    ions: dict[str, arborize.definitions.IonDict]
    mechanisms: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
        dict[str, float]]
    synapses: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
        Union[dict[str, float], arborize.definitions.ExpandedSynapseDict]]

class arborize.definitions.Definition(use_defaults=False)
    Bases: Generic[CT, CP, I, M, S], ABC
    add_cable_type(label: str, def_: CT)
    add_synapse_type(label: str | tuple[str] | tuple[str, str] | tuple[str, str, str], synapse: S)
    abstract class property cable_properties_class: Type[CP]
    abstract class property cable_type_class: Type[CT]
    copy()
    get_cable_types() → dict[str, CT]
    get_synapse_types() → dict[str, S]
    abstract class property ion_class: Type[I]
    abstract class property mechanism_class: Type[M]
```

```

    abstract class property synapse_class: Type[S]

class arborize.definitions.ExpandedSynapseDict
    Bases: dict
    mechanism: str | tuple[str] | tuple[str, str] | tuple[str, str, str]
    parameters: dict[str, float]

class arborize.definitions.Ion(rev_pot: float = None, int_con: float = None, ext_con: float = None)
    Bases: Copy, Merge, Assert, Iterable
    ext_con: float = None
    int_con: float = None
    rev_pot: float = None

class arborize.definitions.IonDict
    Bases: dict
    ext_con: float
    int_con: float
    rev_pot: float

class arborize.definitions.Mechanism(parameters: dict[str, float])
    Bases: object
    copy()
    merge(other)

class arborize.definitions.ModelDefinition(use_defaults=False)
    Bases: Definition[CableType, CableProperties, Ion, Mechanism, Synapse]
    cable_properties_class
        alias of CableProperties
    cable_type_class
        alias of CableType
    ion_class
        alias of Ion
    mechanism_class
        alias of Mechanism
    synapse_class
        alias of Synapse

class arborize.definitions.ModelDefinitionDict
    Bases: dict
    cable_types: dict[str, arborize.definitions.CableTypeDict]
    synapse_types: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
        Union[dict[str, float], arborize.definitions.ExpandedSynapseDict]]

```

```
class arborize.definitions.Synapse(parameters, mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str])
    Bases: Mechanism
    copy()

    mech_id: tuple[str] | tuple[str, str] | tuple[str, str, str]

class arborize.definitions.default_ions_dict(ion_class, *args, **kwargs)
    Bases: dict

arborize.definitions.define_model(template: ModelDefinition, definition: ModelDefinitionDict, /,
                                  use_defaults: bool = False) → ModelDefinition

arborize.definitions.define_model(definition: ModelDefinitionDict, /, use_defaults: bool = False) →
    ModelDefinition

arborize.definitions.is_mech_id(mech_id)

class arborize.definitions.mechdict
    Bases: dict

arborize.definitions.to_mech_id(mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str]) → tuple[str] |
    tuple[str, str] | tuple[str, str, str]
```

5.1.4 arborize.exceptions module

```
exception arborize.exceptions.ArborizeError(*args, **kwargs)
    Bases: DetailedException
    ArborizeError exception

exception arborize.exceptions.ConstructionError(*args, **kwargs)
    Bases: SchematicError
    ConstructionError exception

exception arborize.exceptions.FrozenError(*args, **kwargs)
    Bases: SchematicError
    FrozenError exception

exception arborize.exceptions.ModelDefinitionError(*args, **kwargs)
    Bases: ArborizeError
    ModelDefinitionError exception

exception arborize.exceptions.ModelError(*args, **kwargs)
    Bases: ArborizeError
    ModelError exception

exception arborize.exceptions.SchematicError(*args, **kwargs)
    Bases: ArborizeError
    SchematicError exception

exception arborize.exceptions.TransmitterError(*args, **kwargs)
    Bases: ModelError
    TransmitterError exception
```

exception arborize.exceptions.UnknownLocationError(*args, **kwargs)

Bases: [ModelError](#)

UnknownLocationError exception

exception arborize.exceptions.UnknownSynapseError(*args, **kwargs)

Bases: [ModelError](#)

UnknownSynapseError exception

5.1.5 arborize.parameter module

class arborize.parameter.CableParameter(prop: str, value: float)

Bases: [Parameter](#)

set_cable_params(cable)

class arborize.parameter.IonParameter(ion: str, prop: str, value: float)

Bases: [Parameter](#)

class arborize.parameter.MechParameter

Bases: [Parameter](#)

class arborize.parameter.Parameter

Bases: object

5.1.6 arborize.schematic module

class arborize.schematic.Branch

Bases: object

children: list['Branch']

parent: [Branch](#) | None

points: list[[arborize.schematic.Point](#)]

class arborize.schematic.CableBranch

Bases: [Branch](#)

append(loc, coords, radius, labels)

children: list['CableBranch']

parent: [CableBranch](#) | None

class arborize.schematic.Point(loc, branch: [UnitBranch](#), coords, radius)

Bases: object

class arborize.schematic.Schematic(name=None)

Bases: object

A schematic is an intermediate object that associates parameter definitions to points in space. You can define locations (3d coords + radius) and tag them with labels, or set parameters directly on the locations. You can pass a schematic to a Builder, which will freeze the schematic (no changes can be made anymore) and create a simulator specific instance of the model.

Schematics create a user-facing layer of “virtual branches”, which is the network graph of the created locations. However, NEURON does not support the resolution that arbor does, so an underlying layer of “true branches” is created. In NEURON, a map is kept on the model between the locations on the virtual branches and the locations on the true branches, so that we can arbitrarily split up true branches into smaller pieces to achieve the resolution we need.

arbor: `CableCellTemplate` | `None`

create_empty()

Create an empty branch

create_location(*location: tuple[int, int], coords, radii, labels, endpoint=None*)

Add a new location to the schematic. A location is a tuple of the branch id and point-on-branch id. Locations must be appended in ascending order.

Parameters

- **location** –
- **coords** –
- **radii** –
- **labels** –
- **endpoint** –

Returns

create_name()

Generate the next unique name for an instance of this model.

property definition

Definition of the model, contains the definition of the parameters for the cables, mechanisms, and synapses of this model.

freeze()

Freeze the schematic. Most mutating operations will no longer be permitted.

get_cable_types()

get_compound_cable_types()

get_synapse_types()

property name

Base name for all the instances of this model. Suffixed unique names for each instance can be obtained by calling `create_name`.

set_param(*location: tuple[int, int] | tuple[tuple[int, int], tuple[int, int]] | str, param: [Parameter](#)*)

class `arborize.schematic.UnitBranch`

Bases: [Branch](#)

append(*point*)

children: `list['UnitBranch']`

definition: [CableType](#)

labels: `list[str]`

parent: [UnitBranch](#) | None

`arborize.schematic.throw_frozen()`

5.1.7 arborize.synapse module

class `arborize.synapse.Synapse`(*cell, section, point_process_name, attributes={}, variant=None, type=None, source=None*)

Bases: `object`

presynaptic(*section, x=0.5, **kwargs*)

record()

stimulate(*args, **kwargs)

5.1.8 Module contents

Write descriptions for NEURON cell models in an Arbor-like manner for both the Arbor and NEURON brain simulation engines.

class `arborize.CableProperties`(*Ra: float = None, cm: float = None*)

Bases: `Copy, Merge, Assert, Iterable`

Ra: `float = None`

cm: `float = None`

Axial resistivity in ohm/cm

class `arborize.CableType`(*cable_property_class=<class 'arborize.definitions.CableProperties'>*)

Bases: `object`

add_ion(*key: str, ion: [Ion](#)*)

add_mech(*mech_id: str | tuple[str] | tuple[str, str] | tuple[str, str, str], mech: [Mechanism](#)*)

add_synapse(*label: str | tuple[str] | tuple[str, str] | tuple[str, str, str], synapse: [Synapse](#)*)

classmethod **anchor**(*defs: ~typing.Iterable[~arborize.definitions.CableType], synapses: dict[typing.Union[str, tuple[str], tuple[str, str], tuple[str, str, str]], arborize.definitions.Synapse] | None = None, use_defaults: bool = False, ion_class=<class 'arborize.definitions.Ion'> → [CableType](#)*)

assert_()

cable: [CableProperties](#)

copy()

classmethod **default**(*ion_class=<class 'arborize.definitions.Ion'>*)

ions: `dict[str, arborize.definitions.Ion]`

mechs: `dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]], arborize.definitions.Mechanism]`

```
merge(def_right: CableType)

set(param: Parameter)

synapses: dict[Union[str, tuple[str], tuple[str, str], tuple[str, str, str]],
arborize.definitions.Synapse]

class arborize.Ion(rev_pot: float = None, int_con: float = None, ext_con: float = None)
    Bases: Copy, Merge, Assert, Iterable
    ext_con: float = None
    int_con: float = None
    rev_pot: float = None

class arborize.Mechanism(parameters: dict[str, float])
    Bases: object
    copy()
    merge(other)

class arborize.ModelDefinition(use_defaults=False)
    Bases: Definition[CableType, CableProperties, Ion, Mechanism, Synapse]
    cable_properties_class
        alias of CableProperties
    cable_type_class
        alias of CableType
    ion_class
        alias of Ion
    mechanism_class
        alias of Mechanism
    synapse_class
        alias of Synapse

class arborize.Schematic(name=None)
    Bases: object

    A schematic is an intermediate object that associates parameter definitions to points in space. You can define locations (3d coords + radius) and tag them with labels, or set parameters directly on the locations. You can pass a schematic to a Builder, which will freeze the schematic (no changes can be made anymore) and create a simulator specific instance of the model.

    Schematics create a user-facing layer of “virtual branches”, which is the network graph of the created locations. However, NEURON does not support the resolution that arbor does, so an underlying layer of “true branches” is created. In NEURON, a map is kept on the model between the locations on the virtual branches and the locations on the true branches, so that we can arbitrarily split up true branches into smaller pieces to achieve the resolution we need.

    arbor: CableCellTemplate | None
    cables: list['CableBranch']
```


create_empty()

Create an empty branch

create_location(*location: tuple[int, int], coords, radii, labels, endpoint=None*)

Add a new location to the schematic. A location is a tuple of the branch id and point-on-branch id. Locations must be appended in ascending order.

Parameters

- **location** –
- **coords** –
- **radii** –
- **labels** –
- **endpoint** –

Returns**create_name()**

Generate the next unique name for an instance of this model.

property definition

Definition of the model, contains the definition of the parameters for the cables, mechanisms, and synapses of this model.

freeze()

Freeze the schematic. Most mutating operations will no longer be permitted.

get_cable_types()**get_compound_cable_types()****get_synapse_types()****property name**

Base name for all the instances of this model. Suffixes unique names for each instance can be obtained by calling `create_name`.

roots: list['UnitBranch']

set_param(*location: tuple[int, int] | tuple[tuple[int, int], tuple[int, int]] | str, param: Parameter*)

`arborize.arbor_build`(*schematic: Schematic*)

`arborize.bluepyopt_build`(*schematic: Schematic*)

`arborize.bsb_schematic`(*morphology: Morphology, definitions: Definition | None = None*) → *Schematic*

`arborize.define_constraints`(*constraints: ConstraintsDefinitionDict, tolerance=None, use_defaults=False*) → *ConstraintsDefinition*

`arborize.define_model`(*templ_or_def, def_dict=None, /, use_defaults=False*) → *ModelDefinition*

`arborize.file_schematic`(*file_like: str | os.PathLike | TextIO, definitions: Definition | None = None, fname: str = None, *, name=None*) → *Schematic*

`arborize.is_mech_id`(*mech_id*)

`arborize.neuron_build`(*schematic: Schematic*)

DEFINING A MODEL

Model definitions consist of *cable_types* and *synapse_types*. Cable types define the cable and ionic properties of the types of compartments in the model, such as the soma, axon and dendrites, and which biophysical mechanisms should be present. Synapse types define the properties of the synapses that can be inserted.

```
{
    "cable_types": {...},
    "synapse_types": {...}
}
```

You then pass this dictionary style definition to the `define_model()` function:

```
from arborize import define_model

definition = define_model({
    "cable_types": {...},
    "synapse_types": {...},
})
```

A *definition* can then be combined with a *schematic* by a *builder* to create instances of your model.

6.1 Cable types

Each cable type has a name. A cable type definition consists of *cable*, *ions*, *mechanisms*, and *synapses*. During the schematic construction multiple cable types can be applied to a single piece of cable in the model.

```
{
    "cable_types": {
        "soma": {
            "cable": {...},
            "ions": {...},
            "mechanisms": {...},
            "synapses": {...}
        }
    }
}
```

6.1.1 Cable properties

- Ra: Axial resistivity (ohm/cm)
- cm: Membrane capacitance (uF/cm²)

```
{
  "cable": {
    "Ra": 0.34,
    "cm": 1,
  }
}
```

6.1.2 Ion properties

The *ions* block can contain any key, representing the ion name, conventionally lowercase, and each can set the following properties:

- e: Reversal potential

```
{
  "ions": {
    "h": {"rev_pot": 0},
    "ca": {"rev_pot": 30},
  }
}
```

6.1.3 Mechanisms

Mechanisms are defined by their mechanism ID, which is either a string name, or a tuple of up to 3 strings: name, variant, and package; and a set of parameters.

```
{
  "mechanisms": {
    "Kv1": {
      "gbar": 1.4
    },
    ("Kv1", "burst"): {
      "gbar": 1.4
    }
  }
}
```

Hint: Arborize uses [Glia](#) to manage the mechanisms for the NEURON and Arbor backends. In order to use mechanisms you need to add them to your Glia library.

6.1.4 Synapses

A synapse definition is defined by its name, mechanism ID and parameter set. If you do not specify a mechanism ID, the name is used instead.

```
{
  "synapses": {
    "AMPA": {
      "gmax": 3200
    },
    "depressing": {
      "mechanism": ("AMPA", "TM"),
      "parameters": {
        "gmax": 1300,
        "U": 0.60
      }
    },
    "facilitating": {
      "mechanism": ("AMPA", "TM"),
      "parameters": {
        "gmax": 10000,
        "U": 0.12,
        "tau_facil": 1.1
      }
    }
  },
}
```

6.2 Synapse types

The previous section described how to add synapses to a cable type, but you can also define a synapse type available on the entire model by adding them to the *synapse_types* field.

```
{
  "cable_types": {...},
  "synapse_types": {
    "AMPA": {
      "gmax": 3200
    }
  }
}
```

6.3 Drawing a schematic

Schematics can come from any source, and Arborize supports 2 sources out of the box:

- BSB schematics from BSB morphologies and parameters.
- File schematics load morphologies from file using MorphIO.

```
from arborize import file_schematic, define_model, neuron_build

definition = define_model({
    "cable_types": {...},
    "synapse_types": {...},
})
schematic = file_schematic("my_cell.swc", definition)
n_cells = 100
cells = [neuron_build(schematic) for i in range(n_cells)]
```

PYTHON MODULE INDEX

a

- `arborize`, [19](#)
- `arborize.builders`, [13](#)
- `arborize.definitions`, [13](#)
- `arborize.exceptions`, [16](#)
- `arborize.parameter`, [17](#)
- `arborize.schematic`, [17](#)
- `arborize.schematics`, [13](#)
- `arborize.synapse`, [19](#)

A

add_cable_type() (arborize.definitions.Definition method), 14
 add_ion() (arborize.CableType method), 19
 add_ion() (arborize.definitions.CableType method), 13
 add_mech() (arborize.CableType method), 19
 add_mech() (arborize.definitions.CableType method), 13
 add_synapse() (arborize.CableType method), 19
 add_synapse() (arborize.definitions.CableType method), 13
 add_synapse_type() (arborize.definitions.Definition method), 14
 anchor() (arborize.CableType class method), 19
 anchor() (arborize.definitions.CableType class method), 13
 append() (arborize.schematic.CableBranch method), 17
 append() (arborize.schematic.UnitBranch method), 18
 arbor (arborize.Schematic attribute), 20
 arbor (arborize.schematic.Schematic attribute), 18
 arbor_build() (in module arborize), 21
 arborize
 module, 19
 arborize.builders
 module, 13
 arborize.definitions
 module, 13
 arborize.exceptions
 module, 16
 arborize.parameter
 module, 17
 arborize.schematic
 module, 17
 arborize.schematics
 module, 13
 arborize.synapse
 module, 19
 ArborizeError, 16
 assert_() (arborize.CableType method), 19
 assert_() (arborize.definitions.CableType method), 14

B

bluepyopt_build() (in module arborize), 21

Branch (class in arborize.schematic), 17
 bsb_schematic() (in module arborize), 21

C

cable (arborize.CableType attribute), 19
 cable (arborize.definitions.CableType attribute), 14
 cable (arborize.definitions.CableTypeDict attribute), 14
 cable_properties_class (arborize.definitions.Definition property), 14
 cable_properties_class (arborize.definitions.ModelDefinition attribute), 15
 cable_properties_class (arborize.ModelDefinition attribute), 20
 cable_type_class (arborize.definitions.Definition property), 14
 cable_type_class (arborize.definitions.ModelDefinition attribute), 15
 cable_type_class (arborize.ModelDefinition attribute), 20
 cable_types (arborize.definitions.ModelDefinitionDict attribute), 15
 CableBranch (class in arborize.schematic), 17
 CableParameter (class in arborize.parameter), 17
 CableProperties (class in arborize), 19
 CableProperties (class in arborize.definitions), 13
 CablePropertiesDict (class in arborize.definitions), 13
 cables (arborize.Schematic attribute), 20
 CableType (class in arborize), 19
 CableType (class in arborize.definitions), 13
 CableTypeDict (class in arborize.definitions), 14
 children (arborize.schematic.Branch attribute), 17
 children (arborize.schematic.CableBranch attribute), 17
 children (arborize.schematic.UnitBranch attribute), 18
 cm (arborize.CableProperties attribute), 19
 cm (arborize.definitions.CableProperties attribute), 13
 cm (arborize.definitions.CablePropertiesDict attribute), 13
 ConstructionError, 16

copy() (*arborize.CableType* method), 19
 copy() (*arborize.definitions.CableType* method), 14
 copy() (*arborize.definitions.Definition* method), 14
 copy() (*arborize.definitions.Mechanism* method), 15
 copy() (*arborize.definitions.Synapse* method), 16
 copy() (*arborize.Mechanism* method), 20
 create_empty() (*arborize.Schematic* method), 20
 create_empty() (*arborize.schematic.Schematic* method), 18
 create_location() (*arborize.Schematic* method), 21
 create_location() (*arborize.schematic.Schematic* method), 18
 create_name() (*arborize.Schematic* method), 21
 create_name() (*arborize.schematic.Schematic* method), 18

D

default() (*arborize.CableType* class method), 19
 default() (*arborize.definitions.CableType* class method), 14
 default_ions_dict (class in *arborize.definitions*), 16
 define_constraints() (in module *arborize*), 21
 define_model() (in module *arborize*), 21
 define_model() (in module *arborize.definitions*), 16
 definition (*arborize.Schematic* property), 21
 definition (*arborize.schematic.Schematic* property), 18
 definition (*arborize.schematic.UnitBranch* attribute), 18
 Definition (class in *arborize.definitions*), 14

E

ExpandedSynapseDict (class in *arborize.definitions*), 15
 ext_con (*arborize.definitions.Ion* attribute), 15
 ext_con (*arborize.definitions.IonDict* attribute), 15
 ext_con (*arborize.Ion* attribute), 20

F

file_schematic() (in module *arborize*), 21
 freeze() (*arborize.Schematic* method), 21
 freeze() (*arborize.schematic.Schematic* method), 18
 FrozenError, 16

G

get_cable_types() (*arborize.definitions.Definition* method), 14
 get_cable_types() (*arborize.Schematic* method), 21
 get_cable_types() (*arborize.schematic.Schematic* method), 18
 get_compound_cable_types() (*arborize.Schematic* method), 21
 get_compound_cable_types() (*arborize.schematic.Schematic* method), 18

get_synapse_types() (*arborize.definitions.Definition* method), 14
 get_synapse_types() (*arborize.Schematic* method), 21
 get_synapse_types() (*arborize.schematic.Schematic* method), 18

I

int_con (*arborize.definitions.Ion* attribute), 15
 int_con (*arborize.definitions.IonDict* attribute), 15
 int_con (*arborize.Ion* attribute), 20
 Ion (class in *arborize*), 20
 Ion (class in *arborize.definitions*), 15
 ion_class (*arborize.definitions.Definition* property), 14
 ion_class (*arborize.definitions.ModelDefinition* attribute), 15
 ion_class (*arborize.ModelDefinition* attribute), 20
 IonDict (class in *arborize.definitions*), 15
 IonParameter (class in *arborize.parameter*), 17
 ions (*arborize.CableType* attribute), 19
 ions (*arborize.definitions.CableType* attribute), 14
 ions (*arborize.definitions.CableTypeDict* attribute), 14
 is_mech_id() (in module *arborize*), 21
 is_mech_id() (in module *arborize.definitions*), 16

L

labels (*arborize.schematic.UnitBranch* attribute), 18

M

mech_id (*arborize.definitions.Synapse* attribute), 16
 mechanism (*arborize.definitions.ExpandedSynapseDict* attribute), 15
 Mechanism (class in *arborize*), 20
 Mechanism (class in *arborize.definitions*), 15
 mechanism_class (*arborize.definitions.Definition* property), 14
 mechanism_class (*arborize.definitions.ModelDefinition* attribute), 15
 mechanism_class (*arborize.ModelDefinition* attribute), 20
 mechanisms (*arborize.definitions.CableTypeDict* attribute), 14
 mechdict (class in *arborize.definitions*), 16
 MechParameter (class in *arborize.parameter*), 17
 mechs (*arborize.CableType* attribute), 19
 mechs (*arborize.definitions.CableType* attribute), 14
 merge() (*arborize.CableType* method), 19
 merge() (*arborize.definitions.CableType* method), 14
 merge() (*arborize.definitions.Mechanism* method), 15
 merge() (*arborize.Mechanism* method), 20
 ModelDefinition (class in *arborize*), 20
 ModelDefinition (class in *arborize.definitions*), 15
 ModelDefinitionDict (class in *arborize.definitions*), 15

ModelDefinitionError, 16

ModelError, 16

module

arborize, 19

arborize.builders, 13

arborize.definitions, 13

arborize.exceptions, 16

arborize.parameter, 17

arborize.schematic, 17

arborize.schematics, 13

arborize.synapse, 19

N

name (arborize.Schematic property), 21

name (arborize.schematic.Schematic property), 18

neuron_build() (in module arborize), 21

P

Parameter (class in arborize.parameter), 17

parameters (arborize.definitions.ExpandedSynapseDict attribute), 15

parent (arborize.schematic.Branch attribute), 17

parent (arborize.schematic.CableBranch attribute), 17

parent (arborize.schematic.UnitBranch attribute), 18

Point (class in arborize.schematic), 17

points (arborize.schematic.Branch attribute), 17

presynaptic() (arborize.synapse.Synapse method), 19

R

Ra (arborize.CableProperties attribute), 19

Ra (arborize.definitions.CableProperties attribute), 13

Ra (arborize.definitions.CablePropertiesDict attribute), 13

record() (arborize.synapse.Synapse method), 19

rev_pot (arborize.definitions.Ion attribute), 15

rev_pot (arborize.definitions.IonDict attribute), 15

rev_pot (arborize.Ion attribute), 20

roots (arborize.Schematic attribute), 21

S

Schematic (class in arborize), 20

Schematic (class in arborize.schematic), 17

SchematicError, 16

set() (arborize.CableType method), 20

set() (arborize.definitions.CableType method), 14

set_cable_params() (arborize.parameter.CableParameter method), 17

set_param() (arborize.Schematic method), 21

set_param() (arborize.schematic.Schematic method), 18

stimulate() (arborize.synapse.Synapse method), 19

Synapse (class in arborize.definitions), 15

Synapse (class in arborize.synapse), 19

synapse_class (arborize.definitions.Definition property), 14

synapse_class (arborize.definitions.ModelDefinition attribute), 15

synapse_class (arborize.ModelDefinition attribute), 20

synapse_types (arborize.definitions.ModelDefinitionDict attribute), 15

synapses (arborize.CableType attribute), 20

synapses (arborize.definitions.CableType attribute), 14

synapses (arborize.definitions.CableTypeDict attribute), 14

T

throw_frozen() (in module arborize.schematic), 19

to_mech_id() (in module arborize.definitions), 16

TransmitterError, 16

U

UnitBranch (class in arborize.schematic), 18

UnknownLocationError, 16

UnknownSynapseError, 17