Pyomo Documentation

Release 5.1

Pyomo

Jul 03, 2017

Contents

Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities.

CHAPTER 1

Getting Started

Installation, pyomo command, a simple example, etc

CHAPTER 2

Tutorial

Overview

Pyomo includes a diverse set of optimization capabilities for formulating and analyzing optimization models. Pyomo supports the formulation and analysis of mathematical models for complex optimization applications. This capability is commonly associated with algebraic modeling languages (AMLs), which support the description and analysis of mathematical models with a high-level language. Although most AMLs are implemented in custom modeling languages, Pyomo's modeling objects are embedded within Python, a full-featured high-level programming language that contains a rich set of supporting libraries.

Pyomo has also proven an effective framework for developing high-level optimization and analysis tools. It is easy to develop Python scripts that use Pyomo as a part of a complex analysis workflow. Additionally, Pyomo includes a variety of optimization solvers for stochastic programming, dynamic optimization with differential algebraic equations, mathematical programming with equilibrium conditions, and more! Increasingly, Pyomo is integrating functionality that is normally associated with an optimization solver library.

Concrete vs Abstract Models

Modeling Components

Pyomo Command

CHAPTER 3

Core Pyomo Components

Detailed component descriptions...

CHAPTER 4

Scripting

Scripting examples...

CHAPTER 5

Modeling Extensions

Bilevel Programming

TODO

Dynamic Optimization

TODO

Stochastic Programming

TODO

Generalized Disjunctive Programming

TODO

Stochastic Programming

TODO

CHAPTER 6

Library Reference

Pyomo is being increasingly used as a library to support Python scripts. This section describes library APIs for key elements of Pyomo.

Kernel Library Reference

Low-level Interfaces:

Base Object Storage Interface

class pyomo.core.kernel.component_interface.**IActiveObject**
Bases: object

Interface for objects that support activate/deactivate semantics.

This class is abstract.

activate(**args*, ***kwds*)
Set the active attribute to True

active
A boolean indicating whether or not this object is active.

deactivate(**args*, ***kwds*)
Set the active attribute to False

class pyomo.core.kernel.component_interface.**ICategorizedObject**
Bases: object

Interface for objects that maintain a weak reference to a parent storage object and have a category type.

This class is abstract. It assumes any derived class declares the attributes below at the class or instance level (with or without __slots__):

_ctype
The objects category type.

_parent
A weak reference to the object's parent or `None`.

_is_categorized_object
bool – A flag used to indicate the class is an instance of ICategorized object. This is a workaround for the slow behavior of isinstance on classes that use abc.ABCMeta as a metaclass.

_is_component
bool – A flag used to indicate that the class is an instance of IComponent. This is a workaround for the slow behavior of isinstance on classes that use abc.ABCMeta as a metaclass.

_is_container
bool – A flag used to indicate that the class is an instance IComponentContainer. This is a workaround for the slow behavior of isinstance on classes that use abc.ABCMeta as a metaclass.

getname (*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

root_block
The root storage block above this object

class pyomo.core.kernel.component_interface.**IComponent**
Bases: *pyomo.core.kernel.component_interface.ICategorizedObject*

Interface for components that can be stored inside objects of type IComponentContainer.

This class is abstract, but it partially implements the ICategorizedObject interface by defining the following attributes:

_is_component
True

_is_container
False

class pyomo.core.kernel.component_interface.**IComponentContainer**
Bases: *pyomo.core.kernel.component_interface.ICategorizedObject*

Interface for containers of components or other containers.

This class is abstract, but it partially implements the ICategorizedObject interface by defining the following attributes:

_is_component
False

_is_container
True

child(**args*, ***kwds*)
Returns a child of this container given a storage key.

child_key(**args*, ***kwds*)
Returns the lookup key associated with a child of this container.

children(**args*, ***kwds*)
A generator over the children of this container.

components(**args*, ***kwds*)
A generator over the set of components stored under this container.

postorder_traversal(**args*, ***kwds*)
A generator over all descendents in postfix order.

preorder_traversal(**args*, ***kwds*)
A generator over all descendents in prefix order.

preorder_visit(**args*, ***kwds*)
Visit all descendents in prefix order.

class pyomo.core.kernel.component_interface.**_ActiveComponentContainerMixin**
Bases: *pyomo.core.kernel.component_interface.IActiveObject*

To be used as an additional base class in IComponentContainer implementations to add fuctionality for activating and deactivating the container and its children.

Note: This class is abstract. It assumes any derived class declares the attributes below at the class or instance level (with or without __slots__):

Attributes:

_active (int): A integer that keeps track of the number of active children stored under this container.

_decrement_active()
This method must be called any time an active is child removed or any time an existing child's active status changes from `True` to `False`.

_increment_active()
This method must be called any time a new active child is added or any time an existing child's active status changes from `False` to `True`.

activate(*_from_parent_=False*)
Activate this container. All children of this container will be activated and the active flag on all ancestors of this container will be set to `True`.

active
The active status of this container.

deactivate(*_from_parent_=False*)
Deactivate this container and all of its children.

class pyomo.core.kernel.component_interface.**_ActiveComponentMixin**
Bases: *pyomo.core.kernel.component_interface.IActiveObject*

To be used as an additional base class in IComponent implementations to add fuctionality for activating and deactivating the component.

Any container that stores implementations of this type should use _ActiveComponentContainerMixin as a base class.

This class is abstract. It assumes any derived class declares the attributes below at the class or instance level (with or without __slots__):

_active
bool – A boolean indicating whethor or not this component is active.

activate(*_from_parent_=False*)
Activate this component.

active
The active status of this container.

deactivate(*_from_parent_=False*)
Deactivate this component.

class pyomo.core.kernel.component_interface.**_SimpleContainerMixin**
Bases: `object`

A partial implementation of the IComponentContainer interface for implementations that store a single component category.

Complete implementations need to set the _ctype property at the class level and declare the remaining required abstract properties of the IComponentContainer base class.

Note that this implementation allows nested storage of other *IComponentContainer* implementations that are defined with the same ctype.

_prepare_for_add(*obj*)
This method must be called any time a new child is inserted into this container.

_prepare_for_delete(*obj*)
This method must be called any time a new child is removed from this container.

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)

Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active components should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is `True`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

postorder_traversal(*active=None*, *return_key=False*, *root_key=None*)

Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)

Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

Container Interfaces:

Blocks

class pyomo.core.kernel.component_block.**IBlockStorage**
Bases: *pyomo.core.kernel.component_interface.IComponent*, *pyomo.core.kernel.component_interface.IComponentContainer*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A container that stores multiple types.

This class is abstract, but it partially implements the `ICategorizedObject` interface by defining the following attributes:

_is_component
`True`

_is_container
`True`

activate(*_from_parent_=False*)
Activate this container. All children of this container will be activated and the active flag on all ancestors of this container will be set to `True`.

active
The active status of this container.

child(**args*, ***kwds*)
Returns a child of this container given a storage key.

child_key(**args*, ***kwds*)
Returns the lookup key associated with a child of this container.

clone()
Clones this block. Returns a new block with whose parent pointer is set to `None`. Any components encountered that are descendents of this block will be deepcopied, otherwise a reference to the original component is retained.

deactivate(*_from_parent_=False*)
Deactivate this container and all of its children.

getname(*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

postorder_traversal(**args*, ***kwds*)
A generator over all descendents in postfix order.

preorder_traversal(**args*, ***kwds*)
A generator over all descendents in prefix order.

preorder_visit(**args*, ***kwds*)
Visit all descendents in prefix order.

root_block
The root storage block above this object

class pyomo.core.kernel.component_block.**block**
Bases: pyomo.core.kernel.component_block._block_base, *pyomo.core.kernel.component_block.IBlockStorage*

An implementation of the *IBlockStorage* interface.

activate(*shallow=True*, *descend_into=False*, *_from_parent_=False*)
Activates this block.

Parameters

- **shallow** (*bool*) – If False, all children of the block will be activated. By default, the active status of children are not changed.
- **descend_into** (*bool*) – Indicates whether or not to perform the same action on sub-blocks. The default is False, as a shallow operation on the top-level block is sufficient.

active
The active status of this container.

blocks(*active=None*, *descend_into=True*)
Generates a traversal of all blocks associated with this one (including itself). This method yields identical behavior to calling the components() method with ctype=Block, except that this block is included (as the first item in the generator).

child(*key*)
Get the child object associated with a given storage key for this container.

Raises KeyError – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises ValueError – if the argument is not a child of this container

children(*ctype=<object object>*, *return_key=False*)
Iterate over the children of this block.

Parameters

- **ctype** – Indicate the type of children to iterate over. The default value indicates that all types should be included.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

clone()
Clones this block. Returns a new block with whose parent pointer is set to None. Any components encountered that are descendents of this block will be deepcopied, otherwise a reference to the original component is retained.

collect_ctypes(*active=None*, *descend_into=True*)
Count all object category types stored on or under this block.

Parameters

- **active** (True/None) – Set to True to indicate that only active categorized objects should be counted. The default value of None indicates that all categorized objects (including those that have been deactivated) should be counted. *Note*: This flag is ignored for any objects that do not have an active flag.

- **descend_into** (*bool*) – Indicates whether or not category types should be counted on sub-blocks. Default is `True`.

Returns set of category types

components(*ctype=<object object>*, *active=None*, *return_key=False*, *descend_into=True*)

Generates an efficient traversal of all components stored under this block. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **descend_into** (*bool*) – Indicates whether or not to include components on sub-blocks. Default is `True`.

Returns iterator of objects or (key,object) tuples

deactivate(*shallow=True*, *descend_into=False*, *_from_parent_=False*)

Deactivates this block.

Parameters

- **shallow** (*bool*) – If `False`, all children of the block will be deactivated. By default, the active status of children are not changed, but they become effectively inactive for anything above this block.
- **descend_into** (*bool*) – Indicates whether or not to perform the same action on sub-blocks. The default is `False`, as a shallow operation on the top-level block is sufficient.

generate_names(*ctype=<object object>*, *active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)

Generate a container of fully qualified names (up to this block) for objects stored under this block.

This function is useful in situations where names are used often, but they do not need to be dynamically regenerated each time.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active components should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include components on sub-blocks. Default is `True`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname (*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

load_solution (*solution*, *allow_consistent_values_for_fixed_vars=False*, *comparison_tolerance_for_fixed_vars=1e-05*)
Load a solution.

Parameters

- **solution** – A `pyomo.opt.Solution` object with a symbol map. Optionally, the solution can be tagged with a default variable value (e.g., 0) that will be applied to those variables in the symbol map that do not have a value in the solution.
- **allow_consistent_values_for_fixed_vars** – Indicates whether a solution can specify consistent values for variables that are fixed.
- **comparison_tolerance_for_fixed_vars** – The tolerance used to define whether or not a value in the solution is consistent with the value of a fixed variable.

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

postorder_traversal (*ctype=<object object>*, *active=None*, *include_all_parents=True*, *return_key=False*, *root_key=None*)

Generates a postorder traversal of the storage tree. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the `ctype` keyword is set to something that is not Block. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*ctype=<object object>*, *active=None*, *include_all_parents=True*, *return_key=False*, *root_key=None*)

Generates a preorder traversal of the storage tree. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the `ctype` keyword is set to something that is not Block. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *ctype=<object object>*, *active=None*, *include_all_parents=True*, *include_key=False*, *root_key=None*)

Visits each node in the storage tree using a preorder traversal. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to

True, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the ctype keyword is set to something that is not Block. Default is True.
- **include_key** (*bool*) – Set to True to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when include_key is False.

root_block
The root storage block above this object

write(*filename*, *format=None*, *_solver_capability=None*, *_called_by_solver=False*, ***kwds*)
Write the model to a file, with a given format.

Parameters

- **filename** (*str*) – The name of the file to write.
- **format** – The file format to use. If this is not specified, the file format will be inferred from the filename suffix.
- ****kwds** – Additional keyword options passed to the model writer.

Returns a SymbolMap

class pyomo.core.kernel.component_block.**block_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A dict-style container for blocks.

activate(*_from_parent_=False*)
Activate this container. All children of this container will be activated and the active flag on all ancestors of this container will be set to True.

active
The active status of this container.

child(*key*)
Get the child object associated with a given storage key for this container.

Raises KeyError – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises ValueError – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

clear() → None. Remove all items from D.

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

deactivate(*_from_parent_=False*)
Deactivate this container and all of its children.

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active components should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is `True`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

getname(*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

pop($k[, d]$) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

postorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

root_block
The root storage block above this object

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of D's values

class pyomo.core.kernel.component_block.**block_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A list-style container for blocks.

activate(*_from_parent_=False*)
Activate this container. All children of this container will be activated and the active flag on all ancestors of this container will be set to `True`.

active
The active status of this container.

append(*value*)
S.append(object) – append object to the end of the sequence

child(*key*)
Get the child object associated with a given storage key for this container.

Raises KeyError – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises ValueError – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

count(*value*) → integer – return number of occurrences of value

deactivate(*_from_parent_=False*)
Deactivate this container and all of its children.

extend(*values*)
S.extend(iterable) – extend sequence by appending elements from the iterable

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (True/None) – Set to True to indicate that only active components should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is True.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname (*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert (*i*, *item*)
S.insert(index, object) – insert object before index

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

pop ([*index*]) → item – remove and return item at index (default last).
Raise IndexError if list is empty or index is out of range.

postorder_traversal (*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.

- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

remove(*value*)
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

reverse()
S.reverse() – reverse *IN PLACE*

root_block
The root storage block above this object

class pyomo.core.kernel.component_block.**block_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A tuple-style container for blocks.

activate(*_from_parent_=False*)
Activate this container. All children of this container will be activated and the active flag on all ancestors of this container will be set to `True`.

active
The active status of this container.

child(*key*)
Get the child object associated with a given storage key for this container.

Raises `KeyError` – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises `ValueError` – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

count(*value*) → integer – return number of occurrences of value

deactivate(*_from_parent_=False*)
Deactivate this container and all of its children.

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active components should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is `True`.

- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname (*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is `False`.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

postorder_traversal (*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.

- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

root_block
The root storage block above this object

class pyomo.core.kernel.component_block.**tiny_block**
Bases: pyomo.core.kernel.component_block._block_base, *pyomo.core.kernel.component_block.IBlockStorage*

A memory efficient block for storing a small number of child components.

activate(*shallow=True*, *descend_into=False*, *_from_parent_=False*)
Activates this block.

Parameters

- **shallow** (*bool*) – If `False`, all children of the block will be activated. By default, the active status of children are not changed.
- **descend_into** (*bool*) – Indicates whether or not to perform the same action on sub-blocks. The default is `False`, as a shallow operation on the top-level block is sufficient.

active
The active status of this container.

blocks(*active=None*, *descend_into=True*)
Generates a traversal of all blocks associated with this one (including itself). This method yields identical behavior to calling the components() method with ctype=Block, except that this block is included (as the first item in the generator).

child(*key*)
Get the child object associated with a given storage key for this container.

Raises `KeyError` – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises `ValueError` – if the argument is not a child of this container

children(*ctype=<object object>*, *return_key=False*)
Iterate over the children of this block.

Parameters

- **ctype** – Indicate the type of children to iterate over. The default value indicates that all types should be included.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

clone()
Clones this block. Returns a new block with whose parent pointer is set to `None`. Any components encountered that are descendents of this block will be deepcopied, otherwise a reference to the original component is retained.

collect_ctypes(*active=None*, *descend_into=True*)
Count all object category types stored on or under this block.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active categorized objects should be counted. The default value of `None` indicates that all categorized objects (including those that have been deactivated) should be counted. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not category types should be counted on sub-blocks. Default is `True`.

Returns set of category types

components(*ctype=<object object>*, *active=None*, *return_key=False*, *descend_into=True*)
Generates an efficient traversal of all components stored under this block. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **descend_into** (*bool*) – Indicates whether or not to include components on sub-blocks. Default is True.

Returns iterator of objects or (key,object) tuples

deactivate(*shallow=True*, *descend_into=False*, *_from_parent_=False*)

Deactivates this block.

Parameters

- **shallow** (*bool*) – If False, all children of the block will be deactivated. By default, the active status of children are not changed, but they become effectively inactive for anything above this block.
- **descend_into** (*bool*) – Indicates whether or not to perform the same action on sub-blocks. The default is False, as a shallow operation on the top-level block is sufficient.

generate_names(*ctype=<object object>*, *active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)

Generate a container of fully qualified names (up to this block) for objects stored under this block.

This function is useful in situations where names are used often, but they do not need to be dynamically regenerated each time.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (True/None) – Set to True to indicate that only active components should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include components on sub-blocks. Default is True.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname(*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)

Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is False.

- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

load_solution(*solution*, *allow_consistent_values_for_fixed_vars=False*, *comparison_tolerance_for_fixed_vars=1e-05*)

Load a solution.

Parameters

- **solution** – A `pyomo.opt.Solution` object with a symbol map. Optionally, the solution can be tagged with a default variable value (e.g., 0) that will be applied to those variables in the symbol map that do not have a value in the solution.
- **allow_consistent_values_for_fixed_vars** – Indicates whether a solution can specify consistent values for variables that are fixed.
- **comparison_tolerance_for_fixed_vars** – The tolerance used to define whether or not a value in the solution is consistent with the value of a fixed variable.

local_name

The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name

The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent

The object's parent

parent_block

The first ancestor block above this object

postorder_traversal(*ctype=<object object>*, *active=None*, *include_all_parents=True*, *return_key=False*, *root_key=None*)

Generates a postorder traversal of the storage tree. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that

have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.

- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the `ctype` keyword is set to something that is not Block. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*ctype=<object object>*, *active=None*, *include_all_parents=True*, *return_key=False*, *root_key=None*)

Generates a preorder traversal of the storage tree. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the `ctype` keyword is set to something that is not Block. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *ctype=<object object>*, *active=None*, *include_all_parents=True*, *include_key=False*, *root_key=None*)

Visits each node in the storage tree using a preorder traversal. This includes all components and all component containers (optionally) matching the requested type.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **ctype** – Indicate the type of components to include. The default value indicates that all types should be included.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.

- **include_all_parents** (*bool*) – Indicates if all parent containers (such as blocks and simple block containers) should be included in the traversal even when the ctype keyword is set to something that is not Block. Default is True.
- **include_key** (*bool*) – Set to True to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when include_key is False.

root_block
The root storage block above this object

write(*filename*, *format=None*, *_solver_capability=None*, *_called_by_solver=False*, ***kwds*)
Write the model to a file, with a given format.

Parameters

- **filename** (*str*) – The name of the file to write.
- **format** – The file format to use. If this is not specified, the file format will be inferred from the filename suffix.
- ****kwds** – Additional keyword options passed to the model writer.

Returns a SymbolMap

Tuple-like Object Storage

class pyomo.core.kernel.component_tuple.**ComponentTuple**(**args*)
Bases: *pyomo.core.kernel.component_interface._SimpleContainerMixin*, *pyomo.core.kernel.component_interface.IComponentContainer*, _abcoll.Sequence

A partial implementation of the IComponentContainer interface that presents tuple-like storage functionality.

Complete implementations need to set the _ctype property at the class level, declare the remaining required abstract properties of the IComponentContainer base class, and declare a slot or attribute named _data.

Note that this implementation allows nested storage of other IComponentContainer implementations that are defined with the same ctype.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__metaclass__
alias of ABCMeta

__new__(*S*, ...) → a new object with type S, a subtype of T

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__()
Convert this object to a string by first attempting to generate its fully qualified name. If the object does not have a name (because it does not have a parent, then a string containing the class name is returned.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

__weakref__
list of weak references to the object (if defined)

child(*key*)
Get the child object associated with a given storage key for this container.

Raises KeyError – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises ValueError – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

count(*value*) → integer – return number of occurrences of value

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (True/None) – Set to True to indicate that only active components should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is True.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname (*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is False.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns None.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

postorder_traversal (*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when return_key is False.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)

Generates a preorder traversal of the storage tree.

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when return_key is False.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)

Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the include_key keyword is False, the function signature should be *visit(node) -> [True|False]*. When the include_key keyword is True, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to True, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to True to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when include_key is False.

root_block

The root storage block above this object

List-like Object Storage

class pyomo.core.kernel.component_list.**ComponentList**(**args*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*, _abcoll.MutableSequence

A partial implementation of the IComponentContainer interface that presents list-like storage functionality.

Complete implementations need to set the _ctype property at the class level, declare the remaining required abstract properties of the IComponentContainer base class, and declare a slot or attribute named _data.

Note that this implementation allows nested storage of other IComponentContainer implementations that are defined with the same ctype.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__metaclass__
alias of ABCMeta

__new__(*S*, ...) → a new object with type S, a subtype of T

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__()
Convert this object to a string by first attempting to generate its fully qualified name. If the object does not have a name (because it does not have a parent, then a string containing the class name is returned.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

__weakref__
list of weak references to the object (if defined)

append(*value*)
S.append(object) – append object to the end of the sequence

child(*key*)
Get the child object associated with a given storage key for this container.

Raises KeyError – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises ValueError – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

count(*value*) → integer – return number of occurrences of value

extend(*values*)
S.extend(iterable) – extend sequence by appending elements from the iterable

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (True/None) – Set to True to indicate that only active components should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is True.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

getname(*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is False.

- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns `None`.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert (*i*, *item*)
S.insert(index, object) – insert object before index

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

pop ([*index*]) → item – remove and return item at index (default last).
Raise IndexError if list is empty or index is out of range.

postorder_traversal (*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal (*active=None*, *return_key=False*, *root_key=None*)
Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

remove(*value*)
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

reverse()
S.reverse() – reverse *IN PLACE*

root_block
The root storage block above this object

Dict-like Object Storage

class `pyomo.core.kernel.component_dict.`**ComponentDict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_interface._SimpleContainerMixin*, *pyomo.core.kernel.component_interface.IComponentContainer*, `_abcoll.MutableMapping`

A partial implementation of the IComponentContainer interface that presents dict-like storage functionality.

Complete implementations need to set the _ctype property at the class level, declare the remaining required abstract properties of the IComponentContainer base class, and declare a slot or attribute named _data.

Note that this implementation allows nested storage of other IComponentContainer implementations that are defined with the same ctype.

The optional keyword 'ordered' can be set to `True`/`False` to enable/disable the use of an OrderedDict as the underlying storage dictionary (default is `True`).

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__metaclass__
alias of `ABCMeta`

__new__(*S*, ...) → a new object with type S, a subtype of T

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__()
Convert this object to a string by first attempting to generate its fully qualified name. If the object does not have a name (because it does not have a parent, then a string containing the class name is returned.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

__weakref__
list of weak references to the object (if defined)

child(*key*)
Get the child object associated with a given storage key for this container.

Raises `KeyError` – if the argument is not a storage key for any children of this container

child_key(*child*)
Get the lookup key associated with a child of this container.

Raises `ValueError` – if the argument is not a child of this container

children(*return_key=False*)
Iterate over the children of this container.

Parameters **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the child storage key and the child object. By default, only the child objects are returned.

Returns iterator of objects or (key,object) tuples

clear() → None. Remove all items from D.

components(*active=None*, *return_key=False*)
Generates an efficient traversal of all components stored under this container. Components are leaf nodes in a storage tree (not containers themselves, except for blocks).

Parameters

- **active** (True/None) – Set to True to indicate that only active objects should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to True to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.

Returns iterator of objects or (key,object) tuples

generate_names(*active=None*, *descend_into=True*, *convert=<type 'str'>*, *prefix=''*)
Generate a container of fully qualified names (up to this container) for objects stored under this container.

Parameters

- **active** (True/None) – Set to True to indicate that only active components should be included. The default value of None indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **descend_into** (*bool*) – Indicates whether or not to include subcomponents of any container objects that are not components. Default is True.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is str.
- **prefix** (*str*) – A string to prefix names with.

Returns A component map that behaves as a dictionary mapping component objects to names.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

getname(*fully_qualified=False*, *name_buffer={}*, *convert=<type 'str'>*)
Dynamically generates a name for this object.

Parameters

- **fully_qualified** (*bool*) – Generate a full name by iterating through all anscestor containers. Default is False.
- **convert** (*function*) – A function that converts a storage key into a string representation. Default is the built-in function str.

Returns If a parent exists, this method returns a string representing the name of the object in the context of its parent; otherwise (if no parent exists), this method returns None.

Warning: Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

local_name
The object's local name within the context of its parent. Alias for *obj.getname(fully_qualified=False)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

name
The object's fully qualified name. Alias for *obj.getname(fully_qualified=True)*.

> **Warning:** Name generation can be slow. See the generate_names method, found on most containers, for a way to generate a static set of component names.

parent
The object's parent

parent_block
The first ancestor block above this object

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

postorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a postorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_traversal(*active=None*, *return_key=False*, *root_key=None*)
Generates a preorder traversal of the storage tree.

Parameters

- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the object within its parent and the object itself. By default, only the objects are returned.
- **root_key** – The key to return with this object. Ignored when `return_key` is `False`.

Returns iterator of objects or (key,object) tuples

preorder_visit(*visit*, *active=None*, *include_key=False*, *root_key=None*)
Visits each node in the storage tree using a preorder traversal.

Parameters

- **visit** – A function that is called on each node in the storage tree. When the `include_key` keyword is `False`, the function signature should be *visit(node) -> [True|False]*. When the `include_key` keyword is `True`, the function signature should be *visit(key,node) -> [True|False]*. When the return value of the function evaluates to to `True`, this indicates that the traversal should continue with the children of the current node; otherwise, the traversal does not go below the current node.
- **active** (`True`/`None`) – Set to `True` to indicate that only active objects should be included. The default value of `None` indicates that all components (including those that have been deactivated) should be included. *Note*: This flag is ignored for any objects that do not have an active flag.
- **include_key** (*bool*) – Set to `True` to indicate that 2 arguments should be passed to the visit function, with the first being the local storage key of the object within its parent and the second being the object itself. By default, only the objects are passed to the function.
- **root_key** – The key to pass with this object. Ignored when `include_key` is `False`.

root_block
The root storage block above this object

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of D's values

Modeling Objects:

Variables

Summary

pyomo.core.kernel.component_variable.variable([...])	A decision variable
pyomo.core.kernel.component_variable.variable_tuple(...)	A tuple-style container for variables.
pyomo.core.kernel.component_variable.create_variable_tuple(...)	Generates a full `variable_tuple`.
pyomo.core.kernel.component_variable.variable_list(...)	A list-style container for variables.
pyomo.core.kernel.component_variable.create_variable_list(...)	Generates a full `variable_list`.
pyomo.core.kernel.component_variable.variable_dict(...)	A dict-style container for variables.
pyomo.core.kernel.component_variable.create_variable_dict(...)	Generates a full `variable_dict`.

Member Documentation

class pyomo.core.kernel.component_variable.**variable**(*domain_type=None*, *domain=None*, *lb=None*, *ub=None*, *value=None*, *fixed=False*)

Bases: pyomo.core.kernel.component_variable.IVariable

A decision variable

Decision variables are used in objectives and constraints to define an optimization problem.

Parameters

- **domain_type** – Sets the domain type of the variable. Must be one of RealSet or IntegerSet. Can be updated later by assigning to the domain_type property. The default value of None is equivalent to RealSet, unless the domain keyword is used.
- **domain** – Sets the domain of the variable. This updates the domain_type, lb, and ub properties of the variable. The default value of None implies that this keyword is ignored. This keyword can not be used in combination with the domain_type keyword.
- **lb** – Sets the lower bound of the variable. Can be updated later by assigning to the lb property on the variable. Default is None, which is equivalent to -inf.
- **ub** – Sets the upper bound of the variable. Can be updated later by assigning to the ub property on the variable. Default is None, which is equivalent to +inf.
- **value** – Sets the value of the variable. Can be updated later by assigning to the value property on the variable. Default is None.
- **fixed** (*bool*) – Sets the fixed status of the variable. Can be updated later by assigning to the fixed property or by calling the fix() method. Default is False.

Examples

```
>>> # A continuous variable with infinite bounds
>>> x = pmo.variable()
>>> # A binary variable
>>> x = pmo.variable(domain=pmo.Binary)
>>> # Also a binary variable
>>> x = pmo.variable(domain_type=pmo.IntegerSet, lb=0, ub=1)
```

domain
Set the domain of the variable. This method updates the domain_type property and overwrites the lb and ub properties with the domain bounds.

domain_type
The domain type of the variable (RealSet or IntegerSet)

fixed
The fixed status of the variable

lb
The lower bound of the variable

stale
The stale status of the variable

ub
The upper bound of the variable

value
The value of the variable

class pyomo.core.kernel.component_variable.**variable_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*

A tuple-style container for variables.

pyomo.core.kernel.component_variable.**create_variable_tuple**(*size*, **args*, ***kwds*)
Generates a full variable_tuple.

Parameters

- **size** (*int*) – The number of objects to place in the variable_tuple.
- **type** – The object type to populate the container with. Must have the same ctype as variable_tuple. Default: variable
- ***args** – arguments used to construct the objects placed in the container.
- ****kwds** – keywords used to construct the objects placed in the container.

Returns class:'variable_tuple'

Return type a fully populated

class pyomo.core.kernel.component_variable.**variable_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*

A list-style container for variables.

pyomo.core.kernel.component_variable.**create_variable_list**(*size*, **args*, ***kwds*)
Generates a full variable_list.

Parameters

- **size** (*int*) – The number of objects to place in the variable_list.
- **type** – The object type to populate the container with. Must have the same ctype as variable_list. Default: variable
- ***args** – arguments used to construct the objects placed in the container.
- ****kwds** – keywords used to construct the objects placed in the container.

Returns a fully populated variable_list

class pyomo.core.kernel.component_variable.**variable_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*

A dict-style container for variables.

pyomo.core.kernel.component_variable.**create_variable_dict**(*keys*, **args*, ***kwds*)
Generates a full variable_dict.

Parameters

- **keys** – The set of keys to used to populate the variable_dict.
- **type** – The object type to populate the container with. Must have the same ctype as variable_dict. Default: variable
- ***args** – arguments used to construct the objects placed in the container.
- ****kwds** – keywords used to construct the objects placed in the container.

Returns a fully populated variable_dict

Constraint

Summary

pyomo.core.kernel.component_constraint.constraint([...])	A general algebraic constraint
pyomo.core.kernel.component_constraint.linear_constraint([...])	A linear constraint
pyomo.core.kernel.component_constraint.constraint_tuple(...)	A tuple-style container for constraints.
pyomo.core.kernel.component_constraint.constraint_list(...)	A list-style container for constraints.
pyomo.core.kernel.component_constraint.constraint_dict(...)	A dict-style container for constraints.
pyomo.core.kernel.component_matrix_constraint.matrix_constraint(A)	A container for constraints of the form L <= Ax <= b.

Member Documentation

class pyomo.core.kernel.component_constraint.**constraint**(*expr=None*, *body=None*, *lb=None*, *ub=None*, *rhs=None*)

Bases: pyomo.core.kernel.component_constraint._MutableBoundsConstraintMixin, pyomo.core.kernel.component_constraint.IConstraint

A general algebraic constraint

Algebraic constraints store relational expressions composed of linear or nonlinear functions involving decision variables.

Parameters

- **expr** – Sets the relational expression for the constraint. Can be updated later by assigning to the `expr` property on the constraint. When this keyword is used, values for the `body`, `lb`, `ub`, and `rhs` attributes are automatically determined based on the relational expression type. Default value is `None`.
- **body** – Sets the body of the constraint. Can be updated later by assigning to the `body` property on the constraint. Default is `None`. This keyword should not be used in combination with the `expr` keyword.
- **lb** – Sets the lower bound of the constraint. Can be updated later by assigning to the `lb` property on the constraint. Default is `None`, which is equivalent to `-inf`. This keyword should not be used in combination with the `expr` keyword.
- **ub** – Sets the upper bound of the constraint. Can be updated later by assigning to the `ub` property on the constraint. Default is `None`, which is equivalent to `+inf`. This keyword should not be used in combination with the `expr` keyword.
- **rhs** – Sets the right-hand side of the constraint. Can be updated later by assigning to the `rhs` property on the constraint. The default value of `None` implies that this keyword is ignored. Otherwise, use of this keyword implies that the `equality` property is set to `True`. This keyword should not be used in combination with the `expr` keyword.

Examples

```
>>> # A decision variable used to define constraints
>>> x = pmo.variable()
>>> # An upper bound constraint
>>> c = pmo.constraint(0.5*x <= 1)
>>> # (equivalent form)
>>> c = pmo.constraint(body=0.5*x, ub=1)
>>> # A range constraint
>>> c = pmo.constraint(lb=-1, body=0.5*x, ub=1)
>>> # An nonlinear equality constraint
>>> c = pmo.constraint(x**2 == 1)
>>> # (equivalent form)
>>> c = pmo.constraint(body=x**2, rhs=1)
```

body
The body of the constraint

expr
The full constraint expression –

- lb <= body <= ub: for range constraints
- lb <= body: for lower bounding constraints
- ub >= body: for upper bounding constraints
- body == rhs: for equality constraints

class pyomo.core.kernel.component_constraint.**linear_constraint**(*variables=None*, *coefficients=None*, *terms=None*, *lb=None*, *ub=None*, *rhs=None*)

Bases: pyomo.core.kernel.component_constraint._MutableBoundsConstraintMixin, pyomo.core.kernel.component_constraint.IConstraint

A linear constraint

A linear constraint stores a linear relational expression defined by a list of variables and coefficients. This class can be used to reduce build time and memory for an optimization model. It also increases the speed at which the model can be output to a solver.

Parameters

- **variables** (*list*) – Sets the list of variables in the linear expression defining the body of the constraint. Can be updated later by assigning to the `variables` property on the constraint.
- **coefficients** (*list*) – Sets the list of coefficients for the variables in the linear expression defining the body of the constraint. Can be updated later by assigning to the `coefficients` property on the constraint.
- **terms** (*list*) – An alternative way of initializing the `variables` and `coefficients` lists using an iterable of (variable, coefficient) tuples. Can be updated later by assigning to the `terms` property on the constraint. This keyword should not be used in combination with the `variables` or `coefficients` keywords.
- **lb** – Sets the lower bound of the constraint. Can be updated later by assigning to the `lb` property on the constraint. Default is `None`, which is equivalent to `-inf`.

- **ub** – Sets the upper bound of the constraint. Can be updated later by assigning to the `ub` property on the constraint. Default is `None`, which is equivalent to `+inf`.
- **rhs** – Sets the right-hand side of the constraint. Can be updated later by assigning to the `rhs` property on the constraint. The default value of `None` implies that this keyword is ignored. Otherwise, use of this keyword implies that the `equality` property is set to `True`.

Examples

```
>>> # Decision variables used to define constraints
>>> x = pmo.variable()
>>> y = pmo.variable()
>>> # An upper bound constraint
>>> c = pmo.constraint(variables=[x,y], coefficients=[1,2], ub=1)
>>> # (equivalent form)
>>> c = pmo.constraint(terms=[(x,1), (y,2)], ub=1)
>>> # (equivalent form using a general constraint)
>>> c = pmo.constraint(x + 2*y <= 1)
```

body
The body of the constraint

terms
An iterator over the terms in the body of this constraint as (variable, coefficient) tuples

class pyomo.core.kernel.component_constraint.**constraint_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A tuple-style container for constraints.

class pyomo.core.kernel.component_constraint.**constraint_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A list-style container for constraints.

class pyomo.core.kernel.component_constraint.**constraint_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A dict-style container for constraints.

class pyomo.core.kernel.component_matrix_constraint.**matrix_constraint**(*A*, *lb=None*, *ub=None*, *rhs=None*, *variable_order=None*, *sparse=True*)
Bases: *pyomo.core.kernel.component_constraint.constraint_tuple*

A container for constraints of the form L <= Ax <= b.

Parameters

- **A** – A scipy sparse matrix or 2D numpy array (always copied)

- **lb** – A scalar or array with the same number of rows as A that is set to the lower bound of the constraints
- **ub** – A scalar or array with the same number of rows as A that is set to the upper bound of the constraints
- **rhs** – A scalar or array with the same number of rows as A that is set to the right-hand side the constraints (implies equality constraints)
- **variable_order** – A list with the same number of columns as A that stores the variable associated with each column
- **sparse** – Indicates whether or not sparse storage (CSR format) should be used to store A. Default is `True`.

equality
The array of boolean entries indicating the indices that are equality constraints

lb
The array of constraint lower bounds

lslack
Lower slack (body - lb)

rhs
The array of constraint right-hand sides. Can be set to a scalar or a numpy array of the same dimension. This property can only be read when the equality property is `True` on every index. Assigning to this property implicitly sets the equality property to `True` on every index.

slack
min(lslack, uslack)

sparse
Boolean indicating whether or not the underlying matrix uses sparse storage

ub
The array of constraint upper bounds

uslack
Upper slack (ub - body)

variable_order
The list of variables associated with the columns of the constraint matrix

Parameters

Summary

pyomo.core.kernel.component_parameter.parameter([value])	A placeholder for a mutable, numeric value.
pyomo.core.kernel.component_parameter.parameter_tuple(...)	A tuple-style container for parameters.
pyomo.core.kernel.component_parameter.parameter_list(...)	A list-style container for parameters.
pyomo.core.kernel.component_parameter.parameter_dict(...)	A dict-style container for parameters.

Member Documentation

class pyomo.core.kernel.component_parameter.**parameter**(*value=None*)
Bases: pyomo.core.kernel.component_parameter.IParameter

A placeholder for a mutable, numeric value.

value
The value of the paramater

class pyomo.core.kernel.component_parameter.**parameter_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*

A tuple-style container for parameters.

class pyomo.core.kernel.component_parameter.**parameter_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*

A list-style container for parameters.

class pyomo.core.kernel.component_parameter.**parameter_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*

A dict-style container for parameters.

Objectives

Summary

pyomo.core.kernel.component_objective.objective([...])	An optimization objective.
pyomo.core.kernel.component_objective.objective_tuple(...)	A tuple-style container for objectives.
pyomo.core.kernel.component_objective.objective_list(...)	A list-style container for objectives.
pyomo.core.kernel.component_objective.objective_dict(...)	A dict-style container for objectives.

Member Documentation

class pyomo.core.kernel.component_objective.**objective**(*expr=None*, *sense=1*)
Bases: pyomo.core.kernel.component_objective.IObjective

An optimization objective.

class pyomo.core.kernel.component_objective.**objective_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A tuple-style container for objectives.

class pyomo.core.kernel.component_objective.**objective_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A list-style container for objectives.

class pyomo.core.kernel.component_objective.**objective_dict**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_dict.ComponentDict*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A dict-style container for objectives.

Expressions

Summary

pyomo.core.kernel.component_expression.expression([expr])	A named, mutable expression.
pyomo.core.kernel.component_expression.expression_tuple(...)	A tuple-style container for expressions.
pyomo.core.kernel.component_expression.expression_list(...)	A list-style container for expressions.
pyomo.core.kernel.component_expression.expression_dict(...)	A dict-style container for expressions.

Member Documentation

class pyomo.core.kernel.component_expression.**expression**(*expr=None*)
Bases: pyomo.core.kernel.component_expression.IExpression

A named, mutable expression.

class pyomo.core.kernel.component_expression.**expression_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*

A tuple-style container for expressions.

class pyomo.core.kernel.component_expression.**expression_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*

A list-style container for expressions.

class pyomo.core.kernel.component_expression.**expression_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*

A dict-style container for expressions.

Special Ordered Sets

Summary

pyomo.core.kernel.component_sos.sos(variables)	A Special Ordered Set of type n.
pyomo.core.kernel.component_sos.sos1(variables)	A Special Ordered Set of type 1.
pyomo.core.kernel.component_sos.sos2(variables)	A Special Ordered Set of type 2.
pyomo.core.kernel.component_sos.sos_tuple(...)	A tuple-style container for Special Ordered Sets.

Continued on next page

Table 6.6 – continued from previous page

pyomo.core.kernel.component_sos.sos_list(...)	A list-style container for Special Ordered Sets.
pyomo.core.kernel.component_sos.sos_dict(...)	A dict-style container for Special Ordered Sets.

Member Documentation

class pyomo.core.kernel.component_sos.**sos**(*variables*, *weights=None*, *level=1*)
Bases: pyomo.core.kernel.component_sos.ISOS

A Special Ordered Set of type n.

pyomo.core.kernel.component_sos.**sos1**(*variables*, *weights=None*)
A Special Ordered Set of type 1.

This is an alias for sos(..., level=1)

pyomo.core.kernel.component_sos.**sos2**(*variables*, *weights=None*)
A Special Ordered Set of type 2.

This is an alias for sos(..., level=2).

class pyomo.core.kernel.component_sos.**sos_tuple**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_tuple.ComponentTuple*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A tuple-style container for Special Ordered Sets.

class pyomo.core.kernel.component_sos.**sos_list**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_list.ComponentList*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A list-style container for Special Ordered Sets.

class pyomo.core.kernel.component_sos.**sos_dict**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_dict.ComponentDict*, *pyomo.core.kernel.component_interface._ActiveComponentContainerMixin*

A dict-style container for Special Ordered Sets.

Suffixes

pyomo.core.kernel.component_suffix.**export_suffix_generator**(*blk*, *datatype=<object object>*, *active=None*, *descend_into=True*, *return_key=False*)
Generates an efficient traversal of all suffixes that have been declared for exporting data.

Parameters

- **blk** – A block object.
- **datatype** – Restricts the suffixes included in the returned generator to those matching the provided suffix datatype.
- **active** (True/None) – Set to True to indicate that only active suffixes should be included. The default value of None indicates that all suffixes (including those that have been deactivated) should be included.

- **descend_into** (*bool*) – Indicates whether or not to include suffixes on sub-blocks. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the suffix within its parent and the suffix itself. By default, only the suffixes are returned.

Returns iterator of suffixes or (key,suffix) tuples

pyomo.core.kernel.component_suffix.**import_suffix_generator**(*blk*, *datatype=<object object>*, *active=None*, *descend_into=True*, *return_key=False*)

Generates an efficient traversal of all suffixes that have been declared for importing data.

Parameters

- **blk** – A block object.
- **datatype** – Restricts the suffixes included in the returned generator to those matching the provided suffix datatype.
- **active** (`True`/`None`) – Set to `True` to indicate that only active suffixes should be included. The default value of `None` indicates that all suffixes (including those that have been deactivated) should be included.
- **descend_into** (*bool*) – Indicates whether or not to include suffixes on sub-blocks. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the suffix within its parent and the suffix itself. By default, only the suffixes are returned.

Returns iterator of suffixes or (key,suffix) tuples

pyomo.core.kernel.component_suffix.**local_suffix_generator**(*blk*, *datatype=<object object>*, *active=None*, *descend_into=True*, *return_key=False*)

Generates an efficient traversal of all suffixes that have been declared local data storage.

Parameters

- **blk** – A block object.
- **datatype** – Restricts the suffixes included in the returned generator to those matching the provided suffix datatype.
- **active** (`True`/`None`) – Set to `True` to indicate that only active suffixes should be included. The default value of `None` indicates that all suffixes (including those that have been deactivated) should be included.
- **descend_into** (*bool*) – Indicates whether or not to include suffixes on sub-blocks. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the suffix within its parent and the suffix itself. By default, only the suffixes are returned.

Returns iterator of suffixes or (key,suffix) tuples

class pyomo.core.kernel.component_suffix.**suffix**(**args*, ***kwds*)

Bases: pyomo.core.kernel.component_map.ComponentMap, *pyomo.core.kernel.*

component_interface.IComponent, *pyomo.core.kernel.component_interface._ActiveComponentMixin*

A container for storing extraneous model data that can be imported to or exported from a solver.

datatype
Return the suffix datatype.

direction
Return the suffix direction.

export_enabled
Returns `True` when this suffix is enabled for export to solvers.

import_enabled
Returns `True` when this suffix is enabled for import from solutions.

pyomo.core.kernel.component_suffix.**suffix_generator**(*blk*, *datatype=<object object>*, *active=None*, *descend_into=True*, *return_key=False*)

Generates an efficient traversal of all suffixes that have been declared.

Parameters

- **blk** – A block object.
- **datatype** – Restricts the suffixes included in the returned generator to those matching the provided suffix datatype.
- **active** (`True`/`None`) – Set to `True` to indicate that only active suffixes should be included. The default value of `None` indicates that all suffixes (including those that have been deactivated) should be included.
- **descend_into** (*bool*) – Indicates whether or not to include suffixes on sub-blocks. Default is `True`.
- **return_key** (*bool*) – Set to `True` to indicate that the return type should be a 2-tuple consisting of the local storage key of the suffix within its parent and the suffix itself. By default, only the suffixes are returned.

Returns iterator of suffixes or (key,suffix) tuples

Piecewise Function Library

Modules

Single-variate Piecewise Functions

Summary

pyomo.core.kernel.component_piecewise.transforms.piecewise(...)	Models a single-variate piecewise linear function.
pyomo.core.kernel.component_piecewise.transforms.PiecewiseLinearFunction(...)	A piecewise linear function

Continued on next page

Table 6.7 – continued from previous page

pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction(f)	Base class for transformed piecewise linear functions
pyomo.core.kernel.component_piecewise.transforms.piecewise_convex(...)	Simple convex piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_sos2(...)	Discrete SOS2 piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_dcc(...)	Discrete DCC piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_cc(...)	Discrete CC piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_mc(...)	Discrete MC piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_inc(...)	Discrete INC piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_dlog(...)	Discrete DLOG piecewise representation
pyomo.core.kernel.component_piecewise.transforms.piecewise_log(...)	Discrete LOG piecewise representation

Member Documentation

pyomo.core.kernel.component_piecewise.transforms.**piecewise**(*breakpoints*, *values*, *input=None*, *output=None*, *bound='eq'*, *repn='sos2'*, *validate=True*, *simplify=True*, *equal_slopes_tolerance=1e-06*, *require_bounded_input_variable=True*, *require_variable_domain_coverage=True*)

Models a single-variate piecewise linear function.

This function takes a list breakpoints and function values describing a piecewise linear function and transforms this input data into a block of variables and constraints that enforce a piecewise linear relationship between an input variable and an output variable. In the general case, this transformation requires the use of discrete decision variables.

Parameters

- **breakpoints** (*list*) – The list of breakpoints of the piecewise linear function. This can be a list of numbers or a list of objects that store mutable data (e.g., mutable parameters). If mutable data is used validation might need to be disabled by setting the `validate` keyword to `False`. The list of breakpoints must be in non-decreasing order.
- **values** (*list*) – The values of the piecewise linear function corresponding to the breakpoints.
- **input** – The variable constrained to be the input of the piecewise linear function.
- **output** – The variable constrained to be the output of the piecewise linear function.

- **bound** (*str*) – The type of bound to impose on the output expression. Can be one of:
  - 'lb': y <= f(x)
  - 'eq': y = f(x)
  - 'ub': y >= f(x)
- **repn** (*str*) – The type of piecewise representation to use. Choices are shown below (+ means step functions are supported)
  - 'sos2': standard representation using sos2 constraints (+)
  - 'dcc': disaggregated convex combination (+)
  - 'dlog': logarithmic disaggregated convex combination (+)
  - 'cc': convex combination (+)
  - 'log': logarithmic branching convex combination (+)
  - 'mc': multiple choice
  - 'inc': incremental method (+)
- **validate** (*bool*) – Indicates whether or not to perform validation of the input data. The default is `True`. Validation can be performed manually after the piecewise object is created by calling the `validate()` method. Validation should be performed any time the inputs are changed (e.g., when using mutable parameters in the breakpoints list or when the input variable changes).
- **simplify** (*bool*) – Indicates whether or not to attempt to simplify the piecewise representation to avoid using discrete variables. This can be done when the feasible region for the output variable, with respect to the piecewise function and the bound type, is a convex set. Default is `True`. Validation is required to perform simplification, so this keyword is ignored when the `validate` keyword is `False`.
- **equal_slopes_tolerance** (*float*) – Tolerance used check if consecutive slopes are nearly equal. If any are found, validation will fail. Default is 1e-6. This keyword is ignored when the `validate` keyword is `False`.
- **require_bounded_input_variable** (*bool*) – Indicates if the input variable is required to have finite upper and lower bounds. Default is `True`. Setting this keyword to `False` can be used to allow general expressions to be used as the input in place of a variable. This keyword is ignored when the `validate` keyword is `False`.
- **require_variable_domain_coverage** (*bool*) – Indicates if the function domain (defined by the endpoints of the breakpoints list) needs to cover the entire domain of the input variable. Default is `True`. Ignored for any bounds of variables that are not finite, or when the input is not assigned a variable. This keyword is ignored when the `validate` keyword is `False`.

Returns a block that stores any new variables, constraints, and other components used by the piecewise representation

Return type *TransformedPiecewiseLinearFunction*

class pyomo.core.kernel.component_piecewise.transforms.**PiecewiseLinearFunction**(*breakpoints*, *values*, *validate=True*, ***kwds*)

Bases: object

A piecewise linear function

Piecewise linear functions are defined by a list of breakpoints and a list function values corresponding to each breakpoint. The function value between breakpoints is implied through linear interpolation.

Parameters

- **breakpoints** (*list*) – The list of function breakpoints.
- **values** (*list*) – The list of function values (one for each breakpoint).
- **validate** (*bool*) – Indicates whether or not to perform validation of the input data. The default is True. Validation can be performed manually after the piecewise object is created by calling the validate() method. Validation should be performed any time the inputs are changed (e.g., when using mutable parameters in the breakpoints list).
- ****kwds** – Additional keywords are passed to the validate() method when the validate keyword is True; otherwise, they are ignored.

__call__(*x*)
Evaluates the piecewise linear function at the given point using interpolation

breakpoints
The set of breakpoints used to defined this function

validate(*equal_slopes_tolerance=1e-06*)
Validate this piecewise linear function by verifying various properties of the breakpoints and values lists (e.g., that the list of breakpoints is nondecreasing).

Parameters **equal_slopes_tolerance** (*float*) – Tolerance used check if consecutive slopes are nearly equal. If any are found, validation will fail. Default is 1e-6.

Returns a function characterization code (see util.characterize_function())

Return type int

Raises PiecewiseValidationError – if validation fails

values
The set of values used to defined this function

class pyomo.core.kernel.component_piecewise.transforms.**TransformedPiecewiseLinearFunction**(*f*, *input=None*, *output=None*, *bound='eq'*, *validate=True*, ***kwds*)

Bases: *pyomo.core.kernel.component_block.tiny_block*

Base class for transformed piecewise linear functions

A transformed piecewise linear functions is a block of variables and constraints that enforce a piecewise linear relationship between an input variable and an output variable.

Parameters

- **f** (PiecewiseLinearFunction) – The piecewise linear function to transform.
- **input** – The variable constrained to be the input of the piecewise linear function.

- **output** – The variable constrained to be the output of the piecewise linear function.
- **bound** (*str*) – The type of bound to impose on the output expression. Can be one of:
  - 'lb': y <= f(x)
  - 'eq': y = f(x)
  - 'ub': y >= f(x)
- **validate** (*bool*) – Indicates whether or not to perform validation of the input data. The default is `True`. Validation can be performed manually after the piecewise object is created by calling the `validate()` method. Validation should be performed any time the inputs are changed (e.g., when using mutable parameters in the breakpoints list or when the input variable changes).
- ****kwds** – Additional keywords are passed to the `validate()` method when the `validate` keyword is `True`; otherwise, they are ignored.

__call__(*x*)
Evaluates the piecewise linear function at the given point using interpolation

bound
The bound type assigned to the piecewise relationship ('lb','ub','eq').

breakpoints
The set of breakpoints used to defined this function

input
The expression that stores the input to the piecewise function. The returned object can be updated by assigning to its `expr` attribute.

output
The expression that stores the output of the piecewise function. The returned object can be updated by assigning to its `expr` attribute.

validate(*equal_slopes_tolerance=1e-06*, *require_bounded_input_variable=True*, *require_variable_domain_coverage=True*)
Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

Parameters

- **equal_slopes_tolerance** (*float*) – Tolerance used check if consecutive slopes are nearly equal. If any are found, validation will fail. Default is 1e-6.
- **require_bounded_input_variable** (*bool*) – Indicates if the input variable is required to have finite upper and lower bounds. Default is `True`. Setting this keyword to `False` can be used to allow general expressions to be used as the input in place of a variable.
- **require_variable_domain_coverage** (*bool*) – Indicates if the function domain (defined by the endpoints of the breakpoints list) needs to cover the entire domain of the input variable. Default is `True`. Ignored for any bounds of variables that are not finite, or when the input is not assigned a variable.

Returns a function characterization code (see `util.characterize_function()`)

Return type int

Raises `PiecewiseValidationError` – if validation fails

values
The set of values used to defined this function

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_convex**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Simple convex piecewise representation

Expresses a piecewise linear function with a convex feasible region for the output variable using a simple collection of linear constraints.

validate(***kwds*)

Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_sos2**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete SOS2 piecewise representation

Expresses a piecewise linear function using the SOS2 formulation.

validate(***kwds*)

Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_dcc**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete DCC piecewise representation

Expresses a piecewise linear function using the DCC formulation.

validate(***kwds*)

Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_cc**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete CC piecewise representation

Expresses a piecewise linear function using the CC formulation.

validate(***kwds*)

Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_mc**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete MC piecewise representation

Expresses a piecewise linear function using the MC formulation.

validate(***kwds*)
Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_inc**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete INC piecewise representation

Expresses a piecewise linear function using the INC formulation.

validate(***kwds*)
Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_dlog**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete DLOG piecewise representation

Expresses a piecewise linear function using the DLOG formulation. This formulation uses logarithmic number of discrete variables in terms of number of breakpoints.

validate(***kwds*)
Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

class pyomo.core.kernel.component_piecewise.transforms.**piecewise_log**(**args*, ***kwds*)
Bases: *pyomo.core.kernel.component_piecewise.transforms.TransformedPiecewiseLinearFunction*

Discrete LOG piecewise representation

Expresses a piecewise linear function using the LOG formulation. This formulation uses logarithmic number of discrete variables in terms of number of breakpoints.

validate(***kwds*)
Validate this piecewise linear function by verifying various properties of the breakpoints, values, and input variable (e.g., that the list of breakpoints is nondecreasing).

See base class documentation for keyword descriptions.

Multi-variate Piecewise Functions

Summary

pyomo.core.kernel.component_piecewise.transforms_nd.piecewise_nd(...)	Models a multi-variate piecewise linear function.
pyomo.core.kernel.component_piecewise.transforms_nd.PiecewiseLinearFunctionND(...)	A multi-variate piecewise linear function
pyomo.core.kernel.component_piecewise.transforms_nd.TransformedPiecewiseLinearFunctionND(f)	Base class for transformed multi-variate piecewise
pyomo.core.kernel.component_piecewise.transforms_nd.piecewise_nd_cc(...)	Discrete CC multi-variate piecewise representation

Member Documentation

pyomo.core.kernel.component_piecewise.transforms_nd.**piecewise_nd**(*tri*, *values*, *input=None*, *output=None*, *bound='eq'*, *repn='cc'*)

Models a multi-variate piecewise linear function.

This function takes a D-dimensional triangulation and a list of function values associated with the points of the triangulation and transforms this input data into a block of variables and constraints that enforce a piecewise linear relationship between an D-dimensional vector of input variable and a single output variable. In the general case, this transformation requires the use of discrete decision variables.

Parameters

- **tri** (*scipy.spatial.Delaunay*) – A triangulation over the discretized variable domain. Can be generated using a list of variables using the utility function `util.generate_delaunay()`. Required attributes:
  - points: An (npoints, D) shaped array listing the D-dimensional coordinates of the discretization points.
  - simplices: An (nsimplices, D+1) shaped array of integers specifying the D+1 indices of the points vector that define each simplex of the triangulation.
- **values** (*numpy.array*) – An (npoints,) shaped array of the values of the piecewise function at each of coordinates in the triangulation points array.
- **input** – A D-length list of variables or expressions bound as the inputs of the piecewise function.
- **output** – The variable constrained to be the output of the piecewise linear function.
- **bound** (*str*) – The type of bound to impose on the output expression. Can be one of:
  - 'lb': y <= f(x)
  - 'eq': y = f(x)
  - 'ub': y >= f(x)
- **repn** (*str*) – The type of piecewise representation to use. Can be one of:
  - 'cc': convex combination

Returns a block containing any new variables, constraints, and other components used by the piecewise representation

Return type *TransformedPiecewiseLinearFunctionND*

class pyomo.core.kernel.component_piecewise.transforms_nd.**PiecewiseLinearFunctionND**(*tri*, *values*, *validate=True*, ***kwds*)

Bases: object

A multi-variate piecewise linear function

Multi-varite piecewise linear functions are defined by a triangulation over a finite domain and a list of function values associated with the points of the triangulation. The function value between points in the triangulation is implied through linear interpolation.

Parameters

- **tri** (*scipy.spatial.Delaunay*) – A triangulation over the discretized variable domain. Can be generated using a list of variables using the utility function util.generate_delaunay(). Required attributes:
  - points: An (npoints, D) shaped array listing the D-dimensional coordinates of the discretization points.
  - simplices: An (nsimplices, D+1) shaped array of integers specifying the D+1 indices of the points vector that define each simplex of the triangulation.
- **values** (*numpy.array*) – An (npoints,) shaped array of the values of the piecewise function at each of coordinates in the triangulation points array.

__call__(*x*)

Evaluates the piecewise linear function using interpolation. This method supports vectorized function calls as the interpolation process can be expensive for high dimensional data.

For the case when a single point is provided, the argument x should be a (D,) shaped numpy array or list, where D is the dimension of points in the triangulation.

For the vectorized case, the argument x should be a (n,D)-shaped numpy array.

triangulation

The triangulation over the domain of this function

values

The set of values used to defined this function

class pyomo.core.kernel.component_piecewise.transforms_nd.**TransformedPiecewiseLinearFunctionN**

Bases: *pyomo.core.kernel.component_block.tiny_block*

Base class for transformed multi-variate piecewise linear functions

A transformed multi-variate piecewise linear functions is a block of variables and constraints that enforce a piecewise linear relationship between an vector input variables and a single output variable.

Parameters

- **f** (PiecewiseLinearFunctionND) – The multi-variate piecewise linear function to transform.
- **input** – The variable constrained to be the input of the piecewise linear function.
- **output** – The variable constrained to be the output of the piecewise linear function.
- **bound** (*str*) – The type of bound to impose on the output expression. Can be one of:
  - 'lb': y <= f(x)
  - 'eq': y = f(x)
  - 'ub': y >= f(x)

__call__(*x*)

Evaluates the piecewise linear function using interpolation. This method supports vectorized function calls as the interpolation process can be expensive for high dimensional data.

For the case when a single point is provided, the argument x should be a (D,) shaped numpy array or list, where D is the dimension of points in the triangulation.

For the vectorized case, the argument x should be a (n,D)-shaped numpy array.

bound

The bound type assigned to the piecewise relationship ('lb','ub','eq').

input

The tuple of expressions that store the inputs to the piecewise function. The returned objects can be updated by assigning to their expr attribute.

output

The expression that stores the output of the piecewise function. The returned object can be updated by assigning to its expr attribute.

triangulation

The triangulation over the domain of this function

values

The set of values used to defined this function

class pyomo.core.kernel.component_piecewise.transforms_nd.**piecewise_nd_cc**(**args*, ***kwds*)

Bases: *pyomo.core.kernel.component_piecewise.transforms_nd.TransformedPiecewiseLinearFunctionND*

Discrete CC multi-variate piecewise representation

Expresses a multi-variate piecewise linear function using the CC formulation.

Utilities for Piecewise Functions

exception pyomo.core.kernel.component_piecewise.util.**PiecewiseValidationError**

Bases: exceptions.Exception

An exception raised when validation of piecewise linear functions fail.

pyomo.core.kernel.component_piecewise.util.**characterize_function**(*breakpoints*, *values*)

Characterizes a piecewise linear function described by a list of breakpoints and function values.

Parameters

- **breakpoints** (*list*) – The list of breakpoints of the piecewise linear function. It is assumed that the list of breakpoints is in non-decreasing order.
- **values** (*list*) – The values of the piecewise linear function corresponding to the breakpoints.

Returns a function characterization code and the list of slopes.

Return type (int, list)

Note: The function characterization codes are

- 1: affine
- 2: convex
- 3: concave
- 4: step
- 5: other

If the function has step points, some of the slopes may be `None`.

---

pyomo.core.kernel.component_piecewise.util.**generate_delaunay**(*variables*, *num=10*, ***kwds*)

Generate a Delaunay triangulation of the D-dimensional bounded variable domain given a list of D variables.

Requires numpy and scipy.

Parameters

- **variables** – A list of variables, each having a finite upper and lower bound.
- **num** (*int*) – The number of grid points to generate for each variable (default=10).
- ****kwds** – All additional keywords are passed to the scipy.spatial.Delaunay constructor.

Returns A scipy.spatial.Delaunay object.

pyomo.core.kernel.component_piecewise.util.**generate_gray_code**(*nbits*)

Generates a Gray code of nbits as list of lists

pyomo.core.kernel.component_piecewise.util.**is_constant**(*vals*)

Checks if a list of points is constant

pyomo.core.kernel.component_piecewise.util.**is_nondecreasing**(*vals*)

Checks if a list of points is nondecreasing

pyomo.core.kernel.component_piecewise.util.**is_nonincreasing**(*vals*)

Checks if a list of points is nonincreasing

pyomo.core.kernel.component_piecewise.util.**is_positive_power_of_two**(*x*)

Checks if a number is a nonzero and positive power of 2

pyomo.core.kernel.component_piecewise.util.**log2floor**(*n*)

Computes the exact value of floor(log2(n)) without using floating point calculations. Input argument must be a positive integer.

AML Library Reference

Under construction...

CHAPTER 7

Problem Reference

Examples of Pyomo models for different types of problems ...

CHAPTER 8

Indices and Tables

- genindex
- modindex
- search

CHAPTER 9

Pyomo Resources

The Pyomo home page provides resources for Pyomo users:

- http://pyomo.org

Pyomo development is hosted at GitHub:

- https://github.com/Pyomo/pyomo

See the Pyomo Forum for online discussions of Pyomo:

- http://groups.google.com/group/pyomo-forum/

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

K

L

R

S