API Documentation

API Documentation

June 13, 2007

Contents

1 Package z3c.sqlalchemy

1.1 Modules

- **base** *(Section 2, p. 5)*
- **interfaces** *(Section 3, p. 14)*
- **mapper**: Utility methods for SqlAlchemy
  (Section 4, p. 17)
- **model**: Optional Model support
  (Section 5, p. 24)
- **postgres** *(Section 6, p. 29)*
- **test** *(Section 7, p. 35)*
- **tests** *(Section 8, p. 41)*
  - **testSQLAlchemy**: Tests, tests, tests.........
    (Section 9, p. 42)
- **util**: Some helper methods
  (Section 10, p. 47)

2 Module z3c.sqlalchemy.base

2.1 Variables

Name	Description
session_cache	**Value:** `<z3c.sqlalchemy.base.SynchronizedThreadCache object at 0x...`
connection_cache	**Value:** `<z3c.sqlalchemy.base.SynchronizedThreadCache object at 0x...`

2.2 Class SynchronizedThreadCache

object
  z3c.sqlalchemy.base.SynchronizedThreadCache

2.2.1 Methods

__init__(*self*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

set(*self*, ***kw*)

get(*self*, **names*)

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__new__(*T*, *S*, ...)
Return Value
```
a new object with type S, a subtype of T
```

__reduce__(...)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

2.2.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

2.3 Class BaseWrapper

```
object ─┐
        │
        z3c.sqlalchemy.base.BaseWrapper
```

Known Subclasses: z3c.sqlalchemy.base.ZopeBaseWrapper, z3c.sqlalchemy.postgres.PythonPostgresWrapper

2.3.1 Methods

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(*x*)

hash(x)

__init__(*self*, *dsn*, *model*=`None`, *transactional*=`True`, ***kw*)

```
'dsn' - a RFC-1738-style connection string

'model' - optional instance of model.Model

'kw' - optional keyword arguments passed to create_engine()

'transactional' - True|False, only used by SQLAlchemyDA,
                  *don't touch it*
```

Overrides: object.__init__

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

getMapper(*self*, *tablename*, *schema*=`'public'`)

getMappers(*self*, **names*)

registerMapper(*self*, *mapper*, *name*)

2.3.2 Properties

continued on next page

Name	Description

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`
engine	**Value:** `<property object at 0x2b2433382e60>`
metadata	**Value:** `<property object at 0x2b2433382dc0>`
model	**Value:** `<property object at 0x2b2433382eb0>`
session	**Value:** `<property object at 0x2b2433382e10>`

2.3.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.BaseWrapper>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

2.4 Class SessionDataManager

object ─┐
z3c.sqlalchemy.base.SessionDataManager

Wraps session into transaction context of Zope

2.4.1 Methods

__init__(*self*, *session*, *id*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

abort(*self*, *trans*)

commit(*self*, *trans*)

tpc_begin(*self*, *trans*)

tpc_vote(*self*, *trans*)

tpc_finish(*self*, *trans*)

tpc_abort(*self*, *trans*)

sortKey(*self*)

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(x)

hash(x)

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(x)

repr(x)

__setattr__(*...*)

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

__str__(x)

str(x)

2.4.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

2.4.3 Class Variables

Name	**Description**
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.SessionDataManager>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

2.5 Class ConnectionDataManager

object ─┐
z3c.sqlalchemy.base.ConnectionDataManager

Wraps connection into transaction context of Zope

2.5.1 Methods

__init__(*self*, *connection*, *transactional*=`True`)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

abort(*self*, *trans*)

commit(*self*, *trans*)

tpc_begin(*self*, *trans*)

tpc_vote(*self*, *trans*)

tpc_finish(*self*, *trans*)

tpc_abort(*self*, *trans*)

sortKey(*self*)

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

2.5.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

2.5.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.ConnectionDataManager>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

2.6 Class ZopeBaseWrapper

```
object ─┐
        │
z3c.sqlalchemy.base.BaseWrapper ─┐
                                 │
                                 z3c.sqlalchemy.base.ZopeBaseWrapper
```

Known Subclasses: z3c.sqlalchemy.postgres.ZopePostgresWrapper

A wrapper to be used from within Zope. It connects the session with the transaction management of Zope.

2.6.1 Methods

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__init__(*self*, *dsn*, *model*=`None`, *transactional*=`True`, ***kw*)

```
'dsn' - a RFC-1738-style connection string

'model' - optional instance of model.Model

'kw' - optional keyword arguments passed to create_engine()

'transactional' - True|False, only used by SQLAlchemyDA,
                  *don't touch it*
```

Overrides: object.__init__

__new__(*T*, *S*, ...)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(...)

Object Specification Descriptor

__reduce__(...)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

getMapper(*self*, *tablename*, *schema*=`'public'`)

getMappers(*self*, **names*)

registerMapper(*self*, *mapper*, *name*)

2.6.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`
connection	**Value:** `<property object at 0x2b24333962d0>`
engine	**Value:** `<property object at 0x2b2433382e60>`
metadata	**Value:** `<property object at 0x2b2433382dc0>`
model	**Value:** `<property object at 0x2b2433382eb0>`
session	**Value:** `<property object at 0x2b2433396280>`

2.6.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.BaseWrapper>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

3 Module z3c.sqlalchemy.interfaces

3.1 Class ISQLAlchemyWrapper

zope.interface.Interface

z3c.sqlalchemy.interfaces.ISQLAlchemyWrapper

A SQLAlchemyWrapper wraps sqlalchemy and deals with connection and transaction handling.

3.1.1 Methods

registerMapper(*mapper*, *name*)

register your own mapper under a custom name

getMapper(*tablename*, *schema*=`'public'`)

return a mapper class for a table given by its 'tablename' and an optional 'schema' name

getMappers(**tablenames*)

return a sequence of mapper classes for a given list of table names. ATT: Schema support?

3.1.2 Class Variables

Name	Description
dsn	**Value:** `TextLine(title= u'A RFC-1738 style connection string', re...`
dbname	**Value:** `TextLine(title= u'Database name', required= True)`
host	**Value:** `TextLine(title= u'Hostname of database', required= True)`
port	**Value:** `Int(title= u'Port of database', required= True)`
username	**Value:** `TextLine(title= u'Database user', required= True)`
password	**Value:** `TextLine(title= u'Password of database user', required= T...`
echo	**Value:** `Bool(title= u'Echo all SQL statements to the console', re...`
__bases__	**Value:** `(<InterfaceClass zope.interface.Interface>)`
__identifier__	**Value:** `'z3c.sqlalchemy.interfaces.ISQLAlchemyWrapper'`
__iro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.ISQLAlchemyWra...`
__name__	**Value:** `'ISQLAlchemyWrapper'`
__sro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.ISQLAlchemyWra...`

continued on next page

Name	Description
dependents	**Value:** `<WeakKeyDictionary at 47434478101208>`

3.2 Class IModelProvider

zope.interface.Interface ─┐

z3c.sqlalchemy.interfaces.IModelProvider

A model providers provides information about the tables to be used and the mapper classes.

3.2.1 Methods

getModel(*metadata*=`None`)

The model is described as an ordered dictionary. The entries are (tablename, some_dict) where 'some_dict' is a dictionary containing a key 'table' referencing a Table() instance and an optional key 'relationships' referencing a sequence of related table names. An optional mapper class can be specified through the 'class' key (otherwise a default mapper class will be autogenerated).

3.2.2 Class Variables

Name	Description
__bases__	**Value:** `(<InterfaceClass zope.interface.Interface>)`
__identifier__	**Value:** `'z3c.sqlalchemy.interfaces.IModelProvider'`
__iro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.IModelProvider...`
__name__	**Value:** `'IModelProvider'`
__sro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.IModelProvider...`
dependents	**Value:** `<WeakKeyDictionary at 47434478101136>`

3.3 Class IModel

zope.interface.Interface ─┐

z3c.sqlalchemy.interfaces.IModel

A model represents a configuration hint for SQLAlchemy wrapper instances in order to deliver mappers for a given name.

3.3.1 Methods

add(*name*, *table*=`None`, *mapper_class*=`None`, *relations*=`None`, *autodetect_relations*=`False`, *table_name*=`None`)

'name' – name of table (no schema support so far!)
'table' – a sqlalchemy.Table instance (None, for autoloading)
'mapper_class' – an optional class to be used as mapper class for 'table'
'relations' – an optional list of table names referencing 'table'. This is used for auto-constructing the relation properties of the mapper class.
'autodetect_relations' – try to autodetect the relationships between tables and auto-construct the relation properties of the mapper if 'relations' is omitted (set to None)
'table_name' – optional full name of a table (e.g. 'someschema.sometable') if you want to use 'name' as alias for the table.

items()

return items in insertion order

3.3.2 Class Variables

Name	Description
__bases__	**Value:** `(<InterfaceClass zope.interface.Interface>)`
__identifier__	**Value:** `'z3c.sqlalchemy.interfaces.IModel'`
__iro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.IModel>, <Inte...`
__name__	**Value:** `'IModel'`
__sro__	**Value:** `(<InterfaceClass z3c.sqlalchemy.interfaces.IModel>, <Inte...`
dependents	**Value:** `<WeakKeyDictionary at 47434478101424>`

4 Module z3c.sqlalchemy.mapper

Utility methods for SqlAlchemy

4.1 Class MappedClassBase

object
    z3c.sqlalchemy.mapper.MappedClassBase

Known Subclasses: z3c.sqlalchemy.test.HierarchyNode

base class for all mapped classes

4.1.1 Methods

__init__(*self*, ***kw*)

accepts keywords arguments used for initialization of mapped attributes/columns.

Overrides: object.__init__

clone(*self*)

Create a pristine copy. Use this method if you need to reinsert a copy of the current mapper instance back into the database.

getMapper(*self*, *name*)

Return a mapper associated with the current mapper. If this mapper represents a table A having a relationship to table B then the mapper for B can be obtained through self.getMapper('B'). This method is useful if you don't want to pass
the wrapper around this the wrapper is officially the only way to get hold of a mapper by name. See also http://groups.google.com/group/sqlalchemy/browse_thread/thread/18fb2e2818bdc032/5c2dfd71679925cb#5c2dfd71679925

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(*x*)

hash(x)

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(...)

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

__str__(*x*)

str(x)

4.1.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

4.1.3 Class Variables

Name	Description
__allow_access_to_unprotected-_subobjects__	**Value:** `1`

4.2 Class MapperFactory

object
└ **z3c.sqlalchemy.mapper.MapperFactory**

a factory for table and mapper objects

4.2.1 Methods

__init__(*self*, *metadata*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *table*, *properties*=`{}`, *cls*=`None`)

Returns a tuple (mapped_class, table_class). 'table' - sqlalchemy.Table to be mapped 'properties' - dict containing additional informations about
'cls' - (optional) class used as base for creating the mapper class (will be autogenerated if not available).

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__new__(*T*, *S*, ...)

Return Value

`a new object with type S, a subtype of T`

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(...)

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

__str__(*x*)

str(x)

4.2.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

4.3 Class LazyMapperCollection

```
object ─┐
        │
     dict ─┐
           │
           z3c.sqlalchemy.mapper.LazyMapperCollection
```

Implements a cache for table mappers

4.3.1 Methods

__init__(*self*, *wrapper*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Return Value

    new empty dictionary

Overrides: dict.__init__ extit(inherited documentation)

---

getMapper(*self*, *name*, *schema*=`'public'`)

return a (cached) mapper class for a given table 'name'

__cmp__(*x*, *y*)

cmp(x,y)

__contains__(*D*, *k*)

Return Value

    True if D has a key k, else False

__delattr__(...)

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

__delitem__(*x*, *y*)

del x[y]

__eq__(*x*, *y*)

x==y

__ge__(*x*, *y*)

x>=y

__getattribute__(...)

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

Overrides: object.__getattribute__

__getitem__(*x*, *y*)

x[y]

__gt__(*x*, *y*)

x>y

__hash__(*x*)

hash(x)

Overrides: object.__hash__

__iter__(*x*)

iter(x)

__le__(*x*, *y*)

x<=y

__len__(*x*)

len(x)

__lt__(*x*, *y*)

x<y

__ne__(*x*, *y*)

x!=y

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

Overrides: object.__new__

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

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

__setitem__(*x*, *i*, *y*)

x[i]=y

__str__(*x*)

str(x)

clear(*D*)

Remove all items from D.

Return Value

```
None
```

copy(*D*)

Return Value

```
a shallow copy of D
```

fromkeys(*dict*, *S*, *v*=...)

v defaults to None.

Return Value

```
New dict with keys from S and values equal to v
```

get(*D*, *k*, *d*=...)

d defaults to None.

Return Value

```
D[k] if k in D, else d
```

has_key(*D*, *k*)

Return Value

```
True if D has a key k, else False
```

items(*D*)

Return Value

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

iteritems(*D*)

Return Value

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

iterkeys(*D*)

Return Value

```
an iterator over the keys of D
```

itervalues(*D*)
Return Value
```
an iterator over the values of D
```

keys(*D*)
Return Value
```
list of D's keys
```

pop(*D*, *k*, *d*=...)

If key is not found, d is returned if given, otherwise KeyError is raised

Return Value
```
v, remove specified key and return the corresponding value
```

popitem(*D*)

2-tuple; but raise KeyError if D is empty

Return Value
```
(k, v), remove and return some (key, value) pair as a
```

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

update(*D*, *E*, ***F*)

Update D from E and F: for k in E: D[k] = E[k] (if E has keys else: for (k, v) in E: D[k] = v) then: for k in F: D[k] = F[k]

Return Value
```
None
```

values(*D*)
Return Value
```
list of D's values
```

4.3.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

5 Module z3c.sqlalchemy.model

Optional Model support

5.1 Class Model

```
object ─┐
        │
        dict ─┐
              │
              z3c.sqlalchemy.model.Model
```

The Model is an optional helper class that can be passed to the constructor of a SQLAlchemy wrapper in order to provide hints for the mapper generation.

5.1.1 Methods

__init__(*self*, **args*)

The constructor can be called with a series of dict. Each dict represents a single table and its data (see add() method).

Return Value

`new empty dictionary`

Overrides: dict.__init__

---

add(*self*, *name*, *table*=`None`, *mapper_class*=`None`, *relations*=`None`, *autodetect_relations*=`False`, *table_name*=`None`, *cascade*=`None`)

'name' – name of table (no schema support so far!)
'table' – a sqlalchemy.Table instance (None, for autoloading)
'mapper_class' – an optional class to be used as mapper class for 'table'
'relations' – an optional list of table names referencing 'table'. This is used for auto-constructing the relation properties of the mapper class.
'autodetect_relations' – try to autodetect the relationships between tables and auto-construct the relation properties of the mapper if 'relations' is omitted (set to None)
'table_name' – optional full name of a table (e.g. 'someschema.sometable') if you want to use 'name' as alias for the table.
'cascade' – optional cascade parameter directly passed to the relation() call

---

items(*self*)

return items in insertion order

Return Value

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

Overrides: dict.items

__cmp__(*x*, *y*)

cmp(x,y)

__contains__(*D*, *k*)

Return Value

```
True if D has a key k, else False
```

__delattr__(*...*)

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

__delitem__(*x*, *y*)

del x[y]

__eq__(*x*, *y*)

x==y

__ge__(*x*, *y*)

x>=y

__getattribute__(*...*)

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

Overrides: object.__getattribute__

__getitem__(*x*, *y*)

x[y]

__gt__(*x*, *y*)

x>y

__hash__(*x*)

hash(x)

Overrides: object.__hash__

__iter__(*x*)

iter(x)

__le__(*x*, *y*)

x<=y

__len__(*x*)

len(x)

__lt__(*x*, *y*)

x<y

__ne__(*x*, *y*)

x!=y

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

Overrides: object.__new__

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(*...*)

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

__setitem__(*x*, *i*, *y*)

x[i]=y

__str__(*x*)

str(x)

clear(*D*)

Remove all items from D.

Return Value

```
None
```

copy(*D*)

Return Value

```
a shallow copy of D
```

fromkeys(*dict*, *S*, *v*=...)

v defaults to None.

Return Value

```
New dict with keys from S and values equal to v
```

get(*D*, *k*, *d*=...)

d defaults to None.

Return Value

```
D[k] if k in D, else d
```

has_key(*D*, *k*)

Return Value

```
True if D has a key k, else False
```

iteritems(*D*)

Return Value

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

iterkeys(*D*)

Return Value

```
an iterator over the keys of D
```

itervalues(*D*)

Return Value

```
an iterator over the values of D
```

keys(*D*)

Return Value

```
list of D's keys
```

pop(*D*, *k*, *d*=...)

If key is not found, d is returned if given, otherwise KeyError is raised

Return Value

```
v, remove specified key and return the corresponding value
```

popitem(*D*)

2-tuple; but raise KeyError if D is empty

Return Value

```
(k, v), remove and return some (key, value) pair as a
```

setdefault(*D*, *k*, *d*=...)

Return Value

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

update(*D*, *E*, ***F*)

Update D from E and F: for k in E: D[k] = E[k] (if E has keys else: for (k, v) in E: D[k] = v) then: for k in F: D[k] = F[k]

Return Value

`None`

values(*D*)

Return Value

`list of D’s values`

5.1.2 Properties

Name	Description
__class__	**Value:** `<attribute ’__class__’ of ’object’ objects>`

5.1.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.model.Model>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

6 Module z3c.sqlalchemy.postgres

6.1 Class PostgresMixin

object

z3c.sqlalchemy.postgres.PostgresMixin

Known Subclasses: z3c.sqlalchemy.postgres.PythonPostgresWrapper, z3c.sqlalchemy.postgres.ZopePostgresWrapper

Mixin class for Postgres aspects

6.1.1 Methods

findDependentTables(*self*, *schema*=`'public'`, *ignoreErrors*=`False`)

Returns a mapping tablename -> [list of referencing table(names)]. ATT: this method is specific to Postgres databases! ATT: This method is limited to a particular schema.

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(*x*)

hash(x)

__init__(*...*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(...)

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

__str__(*x*)

str(x)

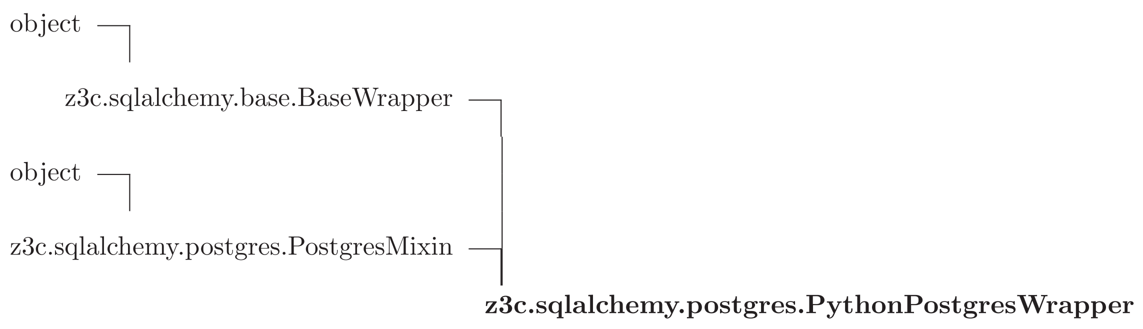
6.1.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

6.1.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.postgres.PostgresMixin>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

6.2 Class PythonPostgresWrapper


object
z3c.sqlalchemy.base.BaseWrapper
object
z3c.sqlalchemy.postgres.PostgresMixin
z3c.sqlalchemy.postgres.PythonPostgresWrapper

Wrapper to be used with Python with extended Postgres functionality.

6.2.1 Methods

__delattr__(...)

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

__getattribute__(*...*)

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

__hash__(*x*)

hash(x)

__init__(*self*, *dsn*, *model*=`None`, *transactional*=`True`, ***kw*)

```
'dsn' - a RFC-1738-style connection string

'model' - optional instance of model.Model

'kw' - optional keyword arguments passed to create_engine()

'transactional' - True|False, only used by SQLAlchemyDA,
                  *don't touch it*
```

Overrides: object.__init__

__new__(*T*, *S*, *...*)

Return Value

```
a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

findDependentTables(*self*, *schema*=`'public'`, *ignoreErrors*=`False`)

Returns a mapping tablename -> [list of referencing table(names)]. ATT: this method is specific to Postgres databases! ATT: This method is limited to a particular schema.

getMapper(*self*, *tablename*, *schema*=`'public'`)

getMappers(*self*, **names*)

registerMapper(*self*, *mapper*, *name*)

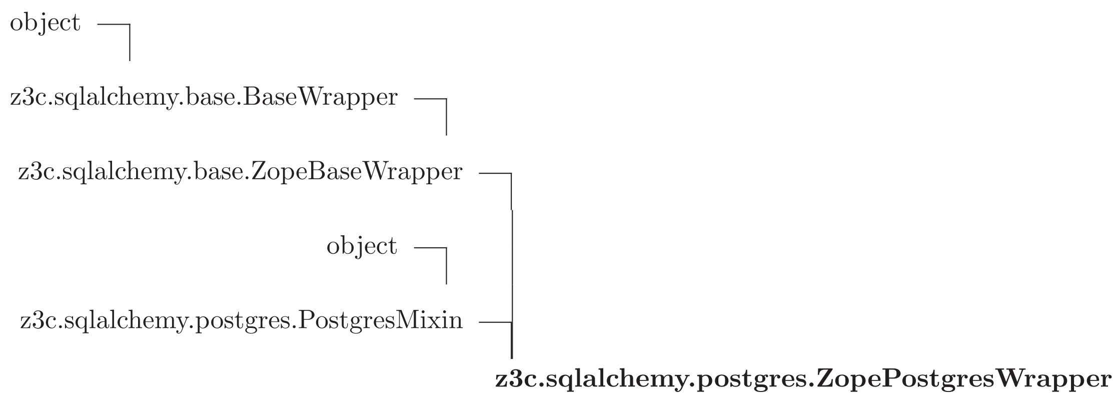
6.2.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`
engine	**Value:** `<property object at 0x2b2433382e60>`
metadata	**Value:** `<property object at 0x2b2433382dc0>`
model	**Value:** `<property object at 0x2b2433382eb0>`
session	**Value:** `<property object at 0x2b2433382e10>`

6.2.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.BaseWrapper>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

6.3 Class ZopePostgresWrapper


object
z3c.sqlalchemy.base.BaseWrapper
z3c.sqlalchemy.base.ZopeBaseWrapper
object
z3c.sqlalchemy.postgres.PostgresMixin
z3c.sqlalchemy.postgres.ZopePostgresWrapper

A wrapper to be used from within Zope. It connects the session with the transaction management of Zope.

6.3.1 Methods

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(*x*)

hash(x)

__init__(*self*, *dsn*, *model*=`None`, *transactional*=`True`, ***kw*)

```
'dsn' - a RFC-1738-style connection string

'model' - optional instance of model.Model

'kw' - optional keyword arguments passed to create_engine()

'transactional' - True|False, only used by SQLAlchemyDA,
                  *don't touch it*
```

Overrides: object.__init__

__new__(*T*, *S*, *...*)

Return Value

```
    a new object with type S, a subtype of T
```

__providedBy__(*...*)

Object Specification Descriptor

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

findDependentTables(*self*, *schema*=`'public'`, *ignoreErrors*=`False`)

Returns a mapping tablename -> [list of referencing table(names)]. ATT: this method is specific to Postgres databases! ATT: This method is limited to a particular schema.

getMapper(*self*, *tablename*, *schema*=`'public'`)

getMappers(*self*, **names*)

registerMapper(*self*, *mapper*, *name*)

6.3.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`
connection	**Value:** `<property object at 0x2b24333962d0>`
engine	**Value:** `<property object at 0x2b2433382e60>`
metadata	**Value:** `<property object at 0x2b2433382dc0>`
model	**Value:** `<property object at 0x2b2433382eb0>`
session	**Value:** `<property object at 0x2b2433396280>`

6.3.3 Class Variables

Name	Description
__implemented__	**Value:** `<implementedBy z3c.sqlalchemy.base.BaseWrapper>`
__provides__	**Value:** `<zope.interface.declarations.ClassProvides object at 0x2b...`

7 Module z3c.sqlalchemy.test

7.1 Variables

Name	Description
dsn	**Value:** `'postgres://postgres:postgres@cmsdb/Toolbox2Test'`
e	**Value:** `create_engine(dsn)`
metadata	**Value:** `BoundMetaData()`
HierarchyTable	**Value:** `Ta-ble('hierarchy',BoundMetaData(),Column(u'id',PGInteger(...`
m	**Value:** `{'hierarchy': {'name': 'hierarchy', 'autodetect_relations...`
wrapper	**Value:** `<z3c.sqlalchemy.postgres.PythonPostgresWrapper object at ...`
session	**Value:** `wrapper.session`
rows	**Value:** `[<z3c.sqlalchemy.test.HierarchyNode object at 0x2b24347ce...`
EXT_PASS	**Value:** `<object object at 0x2b2430628090>`
NULLTYPE	**Value:** `NullTypeEngine()`
default_metadata	**Value:** `DynamicMetaData()`
func	**Value:** `<sqlalchemy.sql._FunctionGenerator object at 0x2b2432c38550>`
modifier	**Value:** `<sqlalchemy.sql._FunctionGenerator object at 0x2b2432c385d0>`

7.2 Class HierarchyNode

```
object ─┐
        │
z3c.sqlalchemy.mapper.MappedClassBase ─┐
                                       │
                                       z3c.sqlalchemy.test.HierarchyNode
```

7.2.1 Methods

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__init__(*self*, **args*, ***kwargs*)

accepts keywords arguments used for initialization of mapped attributes/columns.

Overrides: z3c.sqlalchemy.mapper.MappedClassBase.__init__

__new__(*T*, *S*, *...*)

Return Value

    a new object with type S, a subtype of T

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(*...*)

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

__str__(*x*)

str(x)

clone(*self*)

Create a pristine copy. Use this method if you need to reinsert a copy of the current mapper instance back into the database.

getMapper(*self*, *name*)

Return a mapper associated with the current mapper. If this mapper represents a table A having a relationship to table B then the mapper for B can be obtained through self.getMapper('B'). This method is useful if you don't want to pass
the wrapper around this the wrapper is officially the only way to get hold of a mapper by name. See also http://groups.google.com/group/sqlalchemy/browse_thread/thread/18fb2e2818bdc032/5c2dfd71679925cb#5c2dfd71679925

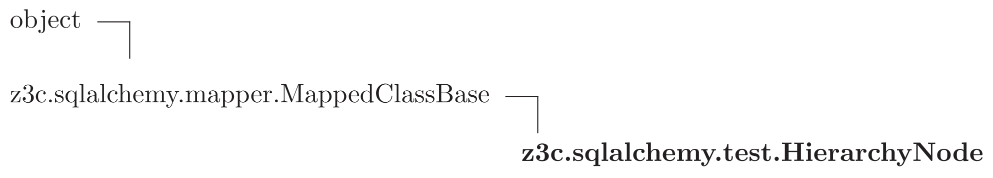
7.2.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

7.2.3 Class Variables

Name	Description
__allow_access_to_unprotected_subobjects__	**Value:** `1`
aedat	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
benutzer	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
bezeichnung	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
c	**Value:** `<sqlalchemy.orm.mapper.LOrderedProp object at 0x2b243355d...`
children	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
comment	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
deleted	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
id	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
idhierarchy_share	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
idprodukt	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
linkindex	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
neudat	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
parent	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
parentid	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
pos	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
produktkuerzel	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
show_gattung_in_bauplan	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
sortierung	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
sorting	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
visible	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`

7.3 Class HierarchyNode


object
z3c.sqlalchemy.mapper.MappedClassBase
z3c.sqlalchemy.test.HierarchyNode

7.3.1 Methods

__delattr__(*...*)

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

__getattribute__(*...*)

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

__hash__(x)

hash(x)

__init__(*self*, **args*, ***kwargs*)

accepts keywords arguments used for initialization of mapped attributes/columns.

Overrides: z3c.sqlalchemy.mapper.MappedClassBase.__init__

__new__(T, S, *...*)
Return Value
```
a new object with type S, a subtype of T
```

__reduce__(*...*)

helper for pickle

__reduce_ex__(*...*)

helper for pickle

__repr__(x)

repr(x)

__setattr__(*...*)

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

__str__(x)

str(x)

clone(*self*)

Create a pristine copy. Use this method if you need to reinsert a copy of the current mapper instance back into the database.

getMapper(*self*, *name*)

Return a mapper associated with the current mapper. If this mapper represents a table A having a relationship to table B then the mapper for B can be obtained through self.getMapper('B'). This method is useful if you don't want to pass
the wrapper around this the wrapper is officially the only way to get hold of a mapper by name. See also http://groups.google.com/group/sqlalchemy/browse_thread/thread/18fb2e2818bdc032/5c2dfd71679925cb#5c2dfd7167992

7.3.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

7.3.3 Class Variables

Name	Description
__allow_access_to_unprotected-_subobjects__	**Value:** `1`
aedat	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
benutzer	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
bezeichnung	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
c	**Value:** `<sqlalchemy.orm.mapper.LOrderedProp object at 0x2b243355d...`
children	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
comment	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
deleted	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
id	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
idhierarchy_share	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
idprodukt	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
linkindex	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
neudat	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`

continued on next page

Name	**Description**
parent	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
parentid	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
pos	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
produktkuerzel	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
show_gattung_in_bauplan	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
sortierung	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
sorting	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`
visible	**Value:** `<sqlalchemy.orm.unitofwork.UOWProperty object at 0x2b2434...`

8 Package z3c.sqlalchemy.tests

8.1 Modules

- **testSQLAlchemy**: Tests, tests, tests.........
  (Section 9, p. 42)

9 Module z3c.sqlalchemy.tests.testSQLAlchemy

Tests, tests, tests.........

9.1 Functions

test_suite()

9.2 Class WrapperTests

object

unittest.TestCase

z3c.sqlalchemy.tests.testSQLAlchemy.WrapperTests

9.2.1 Methods

setUp(*self*)
Hook method for setting up the test fixture before exercising it.

Overrides: unittest.TestCase.setUp extit(inherited documentation)

testIFaceBaseWrapper(*self*)

testIFacePythonPostgres(*self*)

testIFaceZopePostgres(*self*)

testIModel(*self*)

testSimplePopulation(*self*)

testMapperWithCustomModel(*self*)

testCustomMapperClassWithWrongType(*self*)

testGetMappers(*self*)

testModelWeirdParameters(*self*)

testModelWeirdRelationsParameters(*self*)

testModelNonExistingTables(*self*)

testWrapperRegistration(*self*)

testWrapperRegistrationFailing(*self*)

testWrapperDirectRegistration(*self*)

testMapperGetMapper(*self*)

__call__(*self*, **args*, ***kwds*)

__delattr__(...)

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

__getattribute__(...)

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

__hash__(*x*)

hash(x)

__init__(*self*, *methodName*=`'runTest'`)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

Overrides: object.__init__

__new__(*T*, *S*, ...)
Return Value
```
    a new object with type S, a subtype of T
```

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(*self*)
repr(x)

Overrides: object.__repr__ extit(inherited documentation)

__setattr__(...)

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

__str__(*self*)
str(x)

Overrides: object.__str__ extit(inherited documentation)

assertAlmostEqual(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

assertAlmostEquals(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

assertEqual(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are unequal as determined by the '==' operator.

assertEquals(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*self*, *expr*, *msg*=`None`)

Fail the test if the expression is true.

assertNotAlmostEqual(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

assertNotAlmostEquals(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

assertNotEqual(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are equal as determined by the '==' operator.

assertNotEquals(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are equal as determined by the '==' operator.

assertRaises(*self*, *excClass*, *callableObj*, **args*, ***kwargs*)

Fail unless an exception of class excClass is thrown by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

assertTrue(*self*, *expr*, *msg*=`None`)

Fail the test unless the expression is true.

assert_(*self*, *expr*, *msg*=`None`)

Fail the test unless the expression is true.

countTestCases(*self*)

debug(*self*)

Run the test without collecting errors in a TestResult

defaultTestResult(*self*)

fail(*self*, *msg*=`None`)

Fail immediately, with the given message.

failIf(*self*, *expr*, *msg*=`None`)

Fail the test if the expression is true.

failIfAlmostEqual(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

failIfEqual(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are equal as determined by the '==' operator.

failUnless(*self*, *expr*, *msg*=`None`)

Fail the test unless the expression is true.

failUnlessAlmostEqual(*self*, *first*, *second*, *places*=`7`, *msg*=`None`)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero.
Note that decimal places (from zero) are usually not the same as significant digits (measured from the most signficant digit).

failUnlessEqual(*self*, *first*, *second*, *msg*=`None`)

Fail if the two objects are unequal as determined by the '==' operator.

failUnlessRaises(*self*, *excClass*, *callableObj*, **args*, ***kwargs*)

Fail unless an exception of class excClass is thrown by callableObj when invoked with arguments args and keyword arguments kwargs. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

id(*self*)

run(*self*, *result*=`None`)

shortDescription(*self*)

Returns a one-line description of the test, or None if no description has been provided.
The default implementation of this method returns the first line of the specified test method's docstring.

tearDown(*self*)

Hook method for deconstructing the test fixture after testing it.

9.2.2 Properties

Name	Description
__class__	**Value:** `<attribute '__class__' of 'object' objects>`

10 Module z3c.sqlalchemy.util

Some helper methods

10.1 Functions

createSAWrapper(*dsn*, *model*=`None`, *forZope*=`False`, *name*=`None`, *transactional*=`True`, ***kw*)

Convenience method to generate a wrapper for a DSN and a model. This method hides all database related magic from the user.
'dsn' - something like 'postgres://user:password@host/dbname'
'model' - None or an instance of model.Model or a string representing a named utility implementing IModelProvider or a method/callable returning an instance of model.Model.
'forZope' - set this to True in order to obtain a Zope-transaction-aware wrapper.
'transactional' - True|False, only used for SQLAlchemyDA *don't change it*
'name' can be set to register the wrapper automatically in order to avoid a dedicated registerSAWrapper() call.

createSQLAlchemyWrapper(*dsn*, *model*=`None`, *forZope*=`False`, *name*=`None`, *transactional*=`True`, ***kw*)

Convenience method to generate a wrapper for a DSN and a model. This method hides all database related magic from the user.
'dsn' - something like 'postgres://user:password@host/dbname'
'model' - None or an instance of model.Model or a string representing a named utility implementing IModelProvider or a method/callable returning an instance of model.Model.
'forZope' - set this to True in order to obtain a Zope-transaction-aware wrapper.
'transactional' - True|False, only used for SQLAlchemyDA *don't change it*
'name' can be set to register the wrapper automatically in order to avoid a dedicated registerSAWrapper() call.

registerSAWrapper(*wrapper*, *name*)

deferred registration of the wrapper as named utility

registerSQLAlchemyWrapper(*wrapper*, *name*)

deferred registration of the wrapper as named utility

getSAWrapper(*name*)

return a SQLAlchemyWrapper instance by name

getSQLAlchemyWrapper(*name*)

return a SQLAlchemyWrapper instance by name

allRegisteredSAWrappers()

return a dict containing information for all registered wrappers.

allRegisteredSQLAlchemyWrappers()

return a dict containing information for all registered wrappers.

allSAWrapperNames()

return list of all registered wrapper names

Index