



# GeoAlchemy2 Documentation

*Release 0.13.3*

Eric Lemoine

Jun 11, 2023



# CONTENTS

<b>1</b>	<b>Requirements</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>5</b>
<b>3</b>	<b>What's New in GeoAlchemy 2</b>	<b>7</b>
3.1	Migrate to GeoAlchemy 2 . . . . .	7
<b>4</b>	<b>Tutorials</b>	<b>9</b>
4.1	ORM Tutorial . . . . .	9
4.2	Core Tutorial . . . . .	15
4.3	SpatialLite Tutorial . . . . .	21
<b>5</b>	<b>Gallery</b>	<b>27</b>
5.1	Gallery . . . . .	27
<b>6</b>	<b>Use with Alembic</b>	<b>45</b>
6.1	Use Alembic with GeoAlchemy 2 . . . . .	45
<b>7</b>	<b>Reference Documentation</b>	<b>51</b>
7.1	Types . . . . .	51
7.2	Elements . . . . .	55
7.3	Spatial Functions . . . . .	57
7.4	Spatial Operators . . . . .	110
7.5	Shapely Integration . . . . .	113
7.6	Alembic helpers . . . . .	113
<b>8</b>	<b>Development</b>	<b>119</b>
<b>9</b>	<b>Indices and tables</b>	<b>121</b>
	<b>Python Module Index</b>	<b>123</b>
	<b>Index</b>	<b>125</b>



*Using SQLAlchemy with Spatial Databases.*

GeoAlchemy 2 provides extensions to [SQLAlchemy](#) for working with spatial databases.

GeoAlchemy 2 focuses on [PostGIS](#). PostGIS 1.5, PostGIS 2 and PostGIS 3 are supported.

Spatialite is also supported, but using GeoAlchemy 2 with Spatialite requires some specific configuration on the application side. GeoAlchemy 2 works with Spatialite 4.3.0 and higher (except for alembic helpers that need Spatialite  $\geq 5$ ).

GeoAlchemy 2 aims to be simpler than its predecessor, [GeoAlchemy](#). Simpler to use, and simpler to maintain.

The current version of this documentation applies to the version 0.13.3 of GeoAlchemy 2.



## REQUIREMENTS

GeoAlchemy 2 requires [SQLAlchemy](#)  $\geq$  1.4.





## INSTALLATION

GeoAlchemy 2 is [available on the Python Package Index](#). So it can be installed with the standard [pip](#) or [easy\\_install](#) tools.



## WHAT'S NEW IN GEOALCHEMY 2

- GeoAlchemy 2 supports PostGIS' `geometry` type, as well as the `geography` and `raster` types.
- The first series had its own namespace for spatial functions. With GeoAlchemy 2, spatial functions are called like any other SQLAlchemy function, using `func`, which is SQLAlchemy's *standard way* of calling SQL functions.
- GeoAlchemy 2 works with SQLAlchemy's ORM, as well as with SQLAlchemy's *SQL Expression Language* (a.k.a the SQLAlchemy Core). (This is thanks to SQLAlchemy's new *type-level comparator system*.)
- GeoAlchemy 2 supports *reflection* of geometry and geography columns.
- GeoAlchemy 2 adds `to_shape`, `from_shape` functions for a better integration with *Shapely*.

### 3.1 Migrate to GeoAlchemy 2

This section describes how to migrate an application from the first series of GeoAlchemy to GeoAlchemy 2.

#### 3.1.1 Defining Geometry Columns

The first series has specific types like `Point`, `LineString` and `Polygon`. These are gone, the `geoalchemy2.types.Geometry` type should be used instead, and a `geometry_type` can be passed to it.

So, for example, a polygon column that used to be defined like this:

```
geom = Column(Polygon)
```

is now defined like this:

```
geom = Column(Geometry('POLYGON'))
```

This change is related to GeoAlchemy 2 supporting the `geoalchemy2.types.Geography` type.

### 3.1.2 Calling Spatial Functions

The first series has its own namespace/object for calling spatial functions, namely `geoalchemy.functions`. With GeoAlchemy 2, SQLAlchemy's `func` object should be used.

For example, the expression

```
functions.buffer(functions.centroid(box), 10, 2)
```

would be rewritten to this with GeoAlchemy 2:

```
func.ST_Buffer(func.ST_Centroid(box), 10, 2)
```

Also, as the previous example hinted it, the names of spatial functions are now all prefixed with `ST_`. (This is to be consistent with PostGIS and the SQL-MM standard.) The `ST_` prefix should be used even when applying spatial functions to columns, `geoalchemy2.elements.WKTElement`, or `geoalchemy2.elements.WKTElement` objects:

```
Lake.geom.ST_Buffer(10, 2)
lake_table.c.geom.ST_Buffer(10, 2)
lake.geom.ST_Buffer(10, 2)
```

### 3.1.3 WKB and WKT Elements

The first series has classes like `PersistentSpatialElement`, `PGPersistentSpatialElement`, `WKTSpatialElement`.

They're all gone, and replaced by two classes only: `geoalchemy2.elements.WKTElement` and `geoalchemy2.elements.WKBElement`.

`geoalchemy2.elements.WKTElement` is to be used in expressions where a geometry with a specific SRID should be specified. For example:

```
Lake.geom.ST_Touches(WKTElement('POINT(1 1)', srid=4326))
```

If no SRID need be specified, a string can be used directly:

```
Lake.geom.ST_Touches('POINT(1 1)')
```

- `geoalchemy2.elements.WKTElement` literally replaces the first series' `WKTSpatialElement`.
- `geoalchemy2.elements.WKBElement` is the type into which GeoAlchemy 2 converts geometry values read from the database.

For example, the `geom` attributes of `Lake` objects loaded from the database would be references to `geoalchemy2.elements.WKBElement` objects. This class replaces the first series' `PersistentSpatialElement` classes.

See the [Migrate to GeoAlchemy 2](#) page for details on how to migrate a GeoAlchemy application to GeoAlchemy 2.

## TUTORIALS

GeoAlchemy 2 works with both SQLAlchemy's *Object Relational Mapping* (ORM) and *SQL Expression Language*. This documentation provides a tutorial for each system. If you're new to GeoAlchemy 2 start with this.

### 4.1 ORM Tutorial

(This tutorial is greatly inspired by the [SQLAlchemy ORM Tutorial](#), which is recommended reading, eventually.)

GeoAlchemy does not provide an Object Relational Mapper (ORM), but works well with the SQLAlchemy ORM. This tutorial shows how to use the SQLAlchemy ORM with spatial tables, using GeoAlchemy.

#### 4.1.1 Connect to the DB

For this tutorial we will use a PostGIS 2 database. To connect we use SQLAlchemy's `create_engine()` function:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('postgresql://gis:gis@localhost/gis', echo=True)
```

In this example the name of the database, the database user, and the database password, is `gis`.

The `echo` flag is a shortcut to setting up SQLAlchemy logging, which is accomplished via Python's standard logging module. With it is enabled, we'll see all the generated SQL produced.

The return value of `create_engine` is an `Engine` object, which represents the core interface to the database.

#### 4.1.2 Declare a Mapping

When using the ORM, the configurational process starts by describing the database tables we'll be dealing with, and then by defining our own classes which will be mapped to those tables. In modern SQLAlchemy, these two tasks are usually performed together, using a system known as *Declarative*, which allows us to create classes that include directives to describe the actual database table they will be mapped to.

```
>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column, Integer, String
>>> from geoalchemy2 import Geometry
>>>
>>> Base = declarative_base()
>>>
>>> class Lake(Base):
...     __tablename__ = 'lake'
```

(continues on next page)

(continued from previous page)

```
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry('POLYGON'))
```

The Lake class establishes details about the table being mapped, including the name of the table denoted by `__tablename__`, and three columns `id`, `name`, and `geom`. The `id` column will be the primary key of the table. The `geom` column is a `geoalchemy2.types.Geometry` column whose `geometry_type` is `POLYGON`.

### 4.1.3 Create the Table in the Database

The Lake class has a corresponding Table object representing the database table. This Table object was created automatically by SQLAlchemy, it is referenced to by the `Lake.__table__` property:

```
>>> Lake.__table__
Table('lake', MetaData(bind=None), Column('id', Integer(), table=<lake>,
primary_key=True, nullable=False), Column('name', String(), table=<lake>),
Column('geom', Polygon(srid=4326), table=<lake>), schema=None)
```

To create the lake table in the database:

```
>>> Lake.__table__.create(engine)
```

If we wanted to drop the table we'd use:

```
>>> Lake.__table__.drop(engine)
```

### 4.1.4 Create an Instance of the Mapped Class

With the mapping declared, we can create a Lake object:

```
>>> lake = Lake(name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
>>> lake.geom
'POLYGON((0 0,1 0,1 1,0 1,0 0))'
>>> str(lake.id)
'None'
```

A WKT is passed to the Lake constructor for its geometry. This WKT represents the shape of our lake. Since we have not yet told SQLAlchemy to persist the lake object, its `id` is `None`.

The EWKT (Extended WKT) format is also supported. So, for example, if the spatial reference system for the geometry column were 4326, the string `SRID=4326;POLYGON((0 0,1 0,1 1,0 1,0 0))` could be used as the geometry representation.

### 4.1.5 Create a Session

The ORM interacts with the database through a `Session`. Let's create a `Session` class:

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine)
```

This custom-made `Session` class will create new `Session` objects which are bound to our database. Then, whenever we need to have a conversation with the database, we instantiate a `Session`:

```
>>> session = Session()
```

The above `Session` is associated with our PostgreSQL Engine, but it hasn't opened any connection yet.

### 4.1.6 Add New Objects

To persist our Lake object, we add() it to the Session:

```
>>> session.add(lake)
```

At this point the lake object has been added to the `Session`, but no SQL has been issued to the database. The object is in a *pending* state. To persist the object a *flush* or *commit* operation must occur (commit implies flush):

```
>>> session.commit()
```

We can now query the database for Majeur:

[illegible]

`our_lake.geom` is a `geoalchemy2.elements.WKBElement`, which is a type provided by GeoAlchemy. `geoalchemy2.elements.WKBElement` wraps a WKB value returned by the database.

Let's add more lakes:

```
>>> session.add_all([
...     Lake(name='Garde', geom='POLYGON((1 0,3 0,3 2,1 2,1 0))'),
...     Lake(name='Orta', geom='POLYGON((3 0,6 0,6 3,3 3,3 0))')
... ])
>>> session.commit()
```

### 4.1.7 Query

A Query object is created using the `query()` function on Session. For example here's a Query that loads Lake instances ordered by their names:

```
>>> query = session.query(Lake).order_by(Lake.name)
```

Any Query is iterable:

```
>>> for lake in query:
...     print lake.name
...
Garde
Majeur
Orta
```

Another way to execute the query and get a list of Lake objects involves calling `all()` on the Query:

```
>>> lakes = session.query(Lake).order_by(Lake.name).all()
```

The SQLAlchemy ORM Tutorial's [Querying section](#) provides more examples of queries.

### 4.1.8 Make Spatial Queries

Using spatial filters in SQL SELECT queries is very common. Such queries are performed by using spatial relationship functions, or operators, in the `WHERE` clause of the SQL query.

For example, to find the Lake s that contain the point `POINT(4 1)`, we can use this Query:

```
>>> from sqlalchemy import func
>>> query = session.query(Lake).filter(
...     func.ST_Contains(Lake.geom, 'POINT(4 1)'))
...
>>> for lake in query:
...     print lake.name
...
Orta
```

GeoAlchemy allows rewriting this Query more concisely:

```
>>> query = session.query(Lake).filter(Lake.geom.ST_Contains('POINT(4 1)'))
>>> for lake in query:
...     print lake.name
...
Orta
```

Here the `ST_Contains` function is applied to the `Lake.geom` column property. In that case the column property is actually passed to the function, as its first argument.

Here's another spatial filtering query, based on `ST_Intersects`:

```
>>> query = session.query(Lake).filter(
...     Lake.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
...
...

```

(continues on next page)



(continued from previous page)

```
>>> for lake in query:
...     print lake.name
...
Garde
Orta
```

We can also apply relationship functions to `geoalchemy2.elements.WKBElement`. For example:

```
>>> lake = session.query(Lake).filter_by(name='Garde').one()
>>> print session.scalar(lake.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
True
```

`session.scalar` allows executing a clause and returning a scalar value (a boolean value in this case).

The GeoAlchemy functions all start with `ST_`. Operators are also called as functions, but the function names don't include the `ST_` prefix. As an example let's use PostGIS' `&&` operator, which allows testing whether the bounding boxes of geometries intersect. GeoAlchemy provides the `intersects` function for that:

```
>>> query = session.query
>>> query = session.query(Lake).filter(
...     Lake.geom.intersects('LINESTRING(2 1,4 1)'))
...
>>> for lake in query:
...     print lake.name
...
Garde
Orta
```

## 4.1.9 Set Spatial Relationships in the Model

Let's assume that in addition to `lake` we have another table, `treasure`, that includes treasure locations. And let's say that we are interested in discovering the treasures hidden at the bottom of lakes.

The `Treasure` class is the following:

```
>>> class Treasure(Base):
...     __tablename__ = 'treasure'
...     id = Column(Integer, primary_key=True)
...     geom = Column(Geometry('POINT'))
```

We can now add a relationship to the `Lake` table to automatically load the treasures contained by each lake:

```
>>> from sqlalchemy.orm import relationship, backref
>>> class Lake(Base):
...     __tablename__ = 'lake'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry('POLYGON'))
...     treasures = relationship(
...         'Treasure',
...         primaryjoin='func.ST_Contains(foreign(Lake.geom), Treasure.geom).as_
↳ comparison(1, 2)',
```

(continues on next page)

(continued from previous page)

```
...         backref=backref('lake', uselist=False),
...         viewonly=True,
...         uselist=True,
...     )
```

Note the use of the `as_comparison` function. It is required for using an SQL function (`ST_Contains` here) in a `primaryjoin` condition. This only works with SQLAlchemy 1.3, as the `as_comparison` function did not exist before that version. See the [Custom operators based on SQL function](#) section of the SQLAlchemy documentation for more information.

Some information on the parameters used for configuring this relationship:

- `backref` is used to provide the name of property to be placed on the class that handles this relationship in the other direction, namely `Treasure`;
- `viewonly=True` specifies that the relationship is used only for loading objects, and not for persistence operations;
- `uselist=True` indicates that the property should be loaded as a list, as opposed to a scalar.

Also, note that the `treasures` property on `lake` objects (and the `lake` property on `treasure` objects) is loaded “lazily” when the property is first accessed. Another loading strategy may be configured in the `relationship`. For example you’d use `lazy='joined'` for related items to be loaded “eagerly” in the same query as that of the parent, using a `JOIN` or `LEFT OUTER JOIN`.

See the [Relationships API](#) section of the SQLAlchemy documentation for more detail on the `relationship` function, and all the parameters that can be used to configure it.

#### 4.1.10 Use Other Spatial Functions

Here’s a Query that calculates the areas of buffers for our lakes:

```
>>> from sqlalchemy import func
>>> query = session.query(Lake.name,
...                       func.ST_Area(func.ST_Buffer(Lake.geom, 2)) \
...                       .label('bufferarea'))
>>> for row in query:
...     print '%s: %f' % (row.name, row.bufferarea)
...
Majeur: 21.485781
Garde: 32.485781
Orta: 45.485781
```

This Query applies the PostGIS `ST_Buffer` function to the geometry column of every row of the `lake` table. The return value is a list of rows, where each row is actually a tuple of two values: the lake name, and the area of a buffer of the lake. Each tuple is actually an SQLAlchemy `KeyedTuple` object, which provides property type accessors.

Again, the Query can written more concisely:

```
>>> query = session.query(Lake.name,
...                       Lake.geom.ST_Buffer(2).ST_Area().label('bufferarea'))
>>> for row in query:
...     print '%s: %f' % (row.name, row.bufferarea)
...
Majeur: 21.485781
```

(continues on next page)

(continued from previous page)

```
Garde: 32.485781
Orta: 45.485781
```

Obviously, processing and measurement functions can also be used in `WHERE` clauses. For example:

```
>>> lake = session.query(Lake).filter(
...     Lake.geom.ST_Buffer(2).ST_Area() > 33).one()
...
>>> print lake.name
Orta
```

And, like any other functions supported by GeoAlchemy, processing and measurement functions can be applied to `geoalchemy2.elements.WKBElement`. For example:

```
>>> lake = session.query(Lake).filter_by(name='Majeur').one()
>>> bufferarea = session.scalar(lake.geom.ST_Buffer(2).ST_Area())
>>> print '%s: %f' % (lake.name, bufferarea)
Majeur: 21.485781
Majeur: 21.485781
```

#### 4.1.11 Use Raster functions

A few functions (like `ST_Transform()`, `ST_Union()`, `ST_SnapToGrid()`, ...) can be used on both `geoalchemy2.types.Geometry` and `geoalchemy2.types.Raster` types. In GeoAlchemy2, these functions are only defined for `Geometry` as it can not be defined for several types at the same time. Thus using these functions on `Raster` requires minor tweaking to enforce the type by passing the `type_=Raster` argument to the function:

```
>>> query = session.query(Lake.raster.ST_Transform(2154, type_=Raster))
```

#### 4.1.12 Further Reference

- Spatial Functions Reference: *Spatial Functions*
- Spatial Operators Reference: *Spatial Operators*
- Elements Reference: *Elements*

## 4.2 Core Tutorial

(This tutorial is greatly inspired from the [SQLAlchemy SQL Expression Language Tutorial](#), which is recommended reading, eventually.)

This tutorial shows how to use the SQLAlchemy Expression Language (a.k.a. SQLAlchemy Core) with GeoAlchemy. As defined by the SQLAlchemy documentation itself, in contrast to the ORM's domain-centric mode of usage, the SQL Expression Language provides a schema-centric usage paradigm.

### 4.2.1 Connect to the DB

For this tutorial we will use a PostGIS 2 database. To connect we use SQLAlchemy's `create_engine()` function:

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('postgresql://gis:gis@localhost/gis', echo=True)
```

In this example the name of the database, the database user, and the database password, is `gis`.

The `echo` flag is a shortcut to setting up SQLAlchemy logging, which is accomplished via Python's standard logging module. With it is enabled, we'll see all the generated SQL produced.

The return value of `create_engine` is an `Engine` object, which represents the core interface to the database.

### 4.2.2 Define a Table

The very first object that we need to create is a `Table`. Here we create a `lake_table` object, which will correspond to the `lake` table in the database:

```
>>> from sqlalchemy import Table, Column, Integer, String, MetaData
>>> from geoalchemy2 import Geometry
>>>
>>> metadata = MetaData()
>>> lake_table = Table('lake', metadata,
...     Column('id', Integer, primary_key=True),
...     Column('name', String),
...     Column('geom', Geometry('POLYGON'))
... )
```

This table is composed of three columns, `id`, `name` and `geom`. The `geom` column is a `geoalchemy2.types.Geometry` column whose `geometry_type` is `POLYGON`.

Any `Table` object is added to a `MetaData` object, which is a catalog of `Table` objects (and other related objects).

### 4.2.3 Create the Table

With our `Table` being defined we're ready (to have SQLAlchemy) create it in the database:

```
>>> lake_table.create(engine)
```

Calling `create_all()` on `metadata` would have worked equally well:

```
>>> metadata.create_all(engine)
```

In that case every `Table` that's referenced to by `metadata` would be created in the database. The `metadata` object includes one `Table` here, our now well-known `lake_table` object.

## 4.2.4 Reflecting tables

The reflection system of SQLAlchemy can be used on tables containing `geoalchemy2.types.Geometry` or `geoalchemy2.types.Geography` columns. In this case, the type must be imported to be registered into SQLAlchemy, even if it is not used explicitly.

```
>>> from geoalchemy2 import Geometry # <= not used but must be imported
>>> from sqlalchemy import create_engine, MetaData
>>> engine = create_engine("postgresql://myuser:mypass@mydb.host.tld/mydbname")
>>> meta = MetaData()
>>> meta.reflect(bind=engine)
```

## 4.2.5 Insertions

We want to insert records into the lake table. For that we need to create an Insert object. SQLAlchemy provides multiple constructs for creating an Insert object, here's one:

```
>>> ins = lake_table.insert()
>>> str(ins)
INSERT INTO lake (id, name, geom) VALUES (:id, :name, ST_GeomFromEWKT(:geom))
```

The geom column being a Geometry column, the :geom bind value is wrapped in a ST\_GeomFromEWKT call.

To limit the columns named in the INSERT query the values() method can be used:

```
>>> ins = lake_table.insert().values(name='Majeur',
...                                  geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
...
>>> str(ins)
INSERT INTO lake (name, geom) VALUES (:name, ST_GeomFromEWKT(:geom))
```

**Tip:** The string representation of the SQL expression does not include the data placed in values. We got named bind parameters instead. To view the data we can get a compiled form of the expression, and ask for its params:

```
>>> ins.compile.params()
{'geom': 'POLYGON((0 0,1 0,1 1,0 1,0 0))', 'name': 'Majeur'}
```

Up to now we've created an INSERT query but we haven't sent this query to the database yet. Before being able to send it to the database we need a database Connection. We can get a Connection from the Engine object we created earlier:

```
>>> conn = engine.connect()
```

We're now ready to execute our INSERT statement:

```
>>> result = conn.execute(ins)
```

This is what the logging system should output:

```
INSERT INTO lake (name, geom) VALUES (%(name)s, ST_GeomFromEWKT(%(geom)s)) RETURNING
↪ lake.id
```

(continues on next page)

(continued from previous page)

```
{'geom': 'POLYGON((0 0,1 0,1 1,0 1,0 0))', 'name': 'Majeur'}  
COMMIT
```

The value returned by `conn.execute()`, stored in `result`, is a `sqlalchemy.engine.ResultProxy` object. In the case of an `INSERT` we can get the primary key value which was generated from our statement:

```
>>> result.inserted_primary_key  
[1]
```

Instead of using `values()` to specify our `INSERT` data, we can send the data to the `execute()` method on `Connection`. So we could rewrite things as follows:

```
>>> conn.execute(lake_table.insert(),  
...              name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
```

Now let's use another form, allowing to insert multiple rows at once:

```
>>> conn.execute(lake_table.insert(), [  
...     {'name': 'Garde', 'geom': 'POLYGON((1 0,3 0,3 2,1 2,1 0))'},  
...     {'name': 'Orta', 'geom': 'POLYGON((3 0,6 0,6 3,3 3,3 0))'}  
... ])
```

---

**Tip:** In the above examples the geometries are specified as WKT strings. Specifying them as EWKT strings is also supported.

---

## 4.2.6 Selections

Inserting involved creating an `Insert` object, so it'd come to no surprise that Selecting involves creating a `Select` object. The primary construct to generate `SELECT` statements is `SQLAlchemy`'s `select()` function:

```
>>> from sqlalchemy.sql import select  
>>> s = select([lake_table])  
>>> str(s)  
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake
```

The `geom` column being a `Geometry` it is wrapped in a `ST_AsEWKB` call when specified as a column in a `SELECT` statement.

We can now execute the statement and look at the results:

```
>>> result = conn.execute(s)  
>>> for row in result:  
...     print 'name:', row['name'], '; geom:', row['geom'].desc  
...  
name: Majeur ; geom: 0103...  
name: Garde ; geom: 0103...  
name: Orta ; geom: 0103...
```

`row['geom']` is a `geoalchemy2.types.WKBElement` instance. In this example we just get an hexadecimal representation of the geometry's WKB value using the `desc` property.

## 4.2.7 Spatial Query

As spatial database users executing spatial queries is of a great interest to us. There comes GeoAlchemy!

### Spatial relationship

Using spatial filters in SQL SELECT queries is very common. Such queries are performed by using spatial relationship functions, or operators, in the WHERE clause of the SQL query.

For example, to find lakes that contain the point POINT(4 1), we can use this:

```
>>> from sqlalchemy import func
>>> s = select([lake_table],
               func.ST_Contains(lake_table.c.geom, 'POINT(4 1)'))
>>> str(s)
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake WHERE ST_Contains(lake.
↳geom, :param_1)
>>> result = conn.execute(s)
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Orta ; geom: 0103...
```

GeoAlchemy allows rewriting this more concisely:

```
>>> s = select([lake_table], lake_table.c.geom.ST_Contains('POINT(4 1)'))
>>> str(s)
SELECT lake.id, lake.name, ST_AsEWKB(lake.geom) AS geom FROM lake WHERE ST_Contains(lake.
↳geom, :param_1)
```

Here the ST\_Contains function is applied to lake.c.geom. And the generated SQL the lake.geom column is actually passed to the ST\_Contains function as the first argument.

Here's another spatial query, based on ST\_Intersects:

```
>>> s = select([lake_table],
...             lake_table.c.geom.ST_Intersects('LINESTRING(2 1,4 1)'))
>>> result = conn.execute(s)
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Garde ; geom: 0103...
name: Orta ; geom: 0103...
```

This query selects lakes whose geometries intersect ``LINESTRING(2 1,4 1)``.

The GeoAlchemy functions all start with ST\_. Operators are also called as functions, but the names of operator functions don't include the ST\_ prefix.

As an example let's use PostGIS' && operator, which allows testing whether the bounding boxes of geometries intersect. GeoAlchemy provides the intersects function for that:

```
>>> s = select([lake_table],
...             lake_table.c.geom.intersects('LINESTRING(2 1,4 1)'))
>>> result = conn.execute(s)
```

(continues on next page)

(continued from previous page)

```
>>> for row in result:
...     print 'name:', row['name'], '; geom:', row['geom'].desc
...
name: Garde ; geom: 0103...
name: Orta ; geom: 0103...
```

## Processing and Measurement

Here's a Select that calculates the areas of buffers for our lakes:

```
>>> s = select([lake_table.c.name,
                func.ST_Area(
                    lake_table.c.geom.ST_Buffer(2)).label('bufferarea')])
>>> str(s)
SELECT lake.name, ST_Area(ST_Buffer(lake.geom, %(param_1)s)) AS bufferarea FROM lake
>>> result = conn.execute(s)
>>> for row in result:
...     print '%s: %f' % (row['name'], row['bufferarea'])
Majeur: 21.485781
Garde: 32.485781
Orta: 45.485781
```

Obviously, processing and measurement functions can also be used in WHERE clauses. For example:

```
>>> s = select([lake_table.c.name],
                lake_table.c.geom.ST_Buffer(2).ST_Area() > 33)
>>> str(s)
SELECT lake.name FROM lake WHERE ST_Area(ST_Buffer(lake.geom, :param_1)) > :ST_Area_1
>>> result = conn.execute(s)
>>> for row in result:
...     print row['name']
Orta
```

And, like any other functions supported by GeoAlchemy, processing and measurement functions can be applied to *geoalchemy2.elements.WKBElement*. For example:

```
>>> s = select([lake_table], lake_table.c.name == 'Majeur')
>>> result = conn.execute(s)
>>> lake = result.fetchone()
>>> bufferarea = conn.scalar(lake[lake_table.c.geom].ST_Buffer(2).ST_Area())
>>> print '%s: %f' % (lake['name'], bufferarea)
Majeur: 21.485781
```



### 4.2.8 Use Raster functions

A few functions (like `ST_Transform()`, `ST_Union()`, `ST_SnapToGrid()`, ...) can be used on both `geoalchemy2.types.Geometry` and `geoalchemy2.types.Raster` types. In GeoAlchemy2, these functions are only defined for `Geometry` as it can not be defined for several types at the same time. Thus using these functions on `Raster` requires minor tweaking to enforce the type by passing the `type_=Raster` argument to the function:

```
>>> s = select([func.ST_Transform(
    lake_table.c.raster,
    2154,
    type_=Raster)
    .label('transformed_raster')])
```

### 4.2.9 Further Reference

- Spatial Functions Reference: *Spatial Functions*
- Spatial Operators Reference: *Spatial Operators*
- Elements Reference: *Elements*

## 4.3 SpatiaLite Tutorial

GeoAlchemy 2's main target is PostGIS. But GeoAlchemy 2 also supports SpatiaLite, the spatial extension to SQLite. This tutorial describes how to use GeoAlchemy 2 with SpatiaLite. It's based on the *ORM Tutorial*, which you may want to read first.

### 4.3.1 Connect to the DB

Just like when using PostGIS connecting to a SpatiaLite database requires an Engine. This is how you create one for SpatiaLite:

```
>>> from sqlalchemy import create_engine
>>> from sqlalchemy.event import listen
>>>
>>> def load_spatialite(dbapi_conn, connection_record):
...     dbapi_conn.enable_load_extension(True)
...     dbapi_conn.load_extension('/usr/lib/x86_64-linux-gnu/mod_spatialite.so')
...
>>>
>>> engine = create_engine('sqlite:///gis.db', echo=True)
>>> listen(engine, 'connect', load_spatialite)
```

The call to `create_engine` creates an engine bound to the database file `gis.db`. After that a `connect` listener is registered on the engine. The listener is responsible for loading the SpatiaLite extension, which is a necessary operation for using SpatiaLite through SQL.

At this point you can test that you are able to connect to the database:

```
>> conn = engine.connect()
2018-05-30 17:12:02,675 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain_
```

(continues on next page)

(continued from previous page)

```

↳returns' AS VARCHAR(60)) AS anon_1
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine ()
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode_
↳returns' AS VARCHAR(60)) AS anon_1
2018-05-30 17:12:02,676 INFO sqlalchemy.engine.base.Engine ()

```

You can also check that the `gis.db` SQLite database file was created on the file system.

One additional step is required for using Spatialite: create the `geometry_columns` and `spatial_ref_sys` metadata tables. This is done by calling Spatialite's `InitSpatialMetaData` function:

```

>>> from sqlalchemy.sql import select, func
>>>
>>> conn.execute(select([func.InitSpatialMetaData()])))

```

Note that this operation may take some time the first time it is executed for a database. When `InitSpatialMetaData` is executed again it will report an error:

```
InitSpatialMetaData() error:"table spatial_ref_sys already exists"
```

You can safely ignore that error.

Before going further we can close the current connection:

```
>>> conn.close()
```

### 4.3.2 Declare a Mapping

Now that we have a working connection we can go ahead and create a mapping between a Python class and a database table.

```

>>> from sqlalchemy.ext.declarative import declarative_base
>>> from sqlalchemy import Column, Integer, String
>>> from geoalchemy2 import Geometry
>>>
>>> Base = declarative_base()
>>>
>>> class Lake(Base):
...     __tablename__ = 'lake'
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     geom = Column(Geometry(geometry_type='POLYGON', management=True))

```

This basically works in the way as with PostGIS. The difference is the `management` argument that must be set to `True`.

Setting `management` to `True` indicates that the `AddGeometryColumn` and `DiscardGeometryColumn` management functions will be used for the creation and removal of the geometry column. This is required with Spatialite.

### 4.3.3 Create the Table in the Database

We can now create the lake table in the `gis.db` database:

```
>>> Lake.__table__.create(engine)
```

If we wanted to drop the table we'd use:

```
>>> Lake.__table__.drop(engine)
```

There's nothing specific to SpatialLite here.

### 4.3.4 Create a Session

When using the SQLAlchemy ORM the ORM interacts with the database through a `Session`.

```
>>> from sqlalchemy.orm import sessionmaker
>>> Session = sessionmaker(bind=engine)
>>> session = Session()
```

The session is associated with our SpatiaLite Engine. Again, there's nothing specific to SpatiaLite here.

### 4.3.5 Add New Objects

We can now create and insert new Lake objects into the database, the same way we'd do it using GeoAlchemy 2 with PostGIS.

```
>>> lake = Lake(name='Majeur', geom='POLYGON((0 0,1 0,1 1,0 1,0 0))')
>>> session.add(lake)
>>> session.commit()
```

We can now query the database for Majeur:

[illegible]

Let's add more lakes:

```
>>> session.add_all([
...     Lake(name='Garde', geom='POLYGON((1 0,3 0,3 2,1 2,1 0))'),
...     Lake(name='Orta', geom='POLYGON((3 0,6 0,6 3,3 3,3 0))')
... ])
>>> session.commit()
```

### 4.3.6 Query

Let's make a simple, non-spatial, query:

```
>>> query = session.query(Lake).order_by(Lake.name)
>>> for lake in query:
...     print(lake.name)
...
Garde
Majeur
Orta
```

Now a spatial query:

```
>>> from geolalchemy2 import WKTElement
>>> query = session.query(Lake).filter(
...     func.ST_Contains(Lake.geom, WKTElement('POINT(4 1)'))
... )
>>> for lake in query:
...     print(lake.name)
...
Orta
```

Here's another spatial query, using ST\_Intersects this time:

```
>>> query = session.query(Lake).filter(
...     Lake.geom.ST_Intersects(WKTElement('LINESTRING(2 1,4 1)'))
... )
>>> for lake in query:
...     print(lake.name)
...
Garde
Orta
```

We can also apply relationship functions to *geoalchemy2.elements.WKBElement*. For example:

```
>>> lake = session.query(Lake).filter_by(name='Garde').one()
>>> print(session.scalar(lake.geom.ST_Intersects(WKTElement('LINESTRING(2 1,4 1)'))))
1
```

`session.scalar` allows executing a clause and returning a scalar value (an integer value in this case).

The value 1 indicates that the lake “Garde” does intersects the `LINESTRING(2 1,4 1)` geometry. See the Spatialite SQL functions reference list for more information.

### 4.3.7 Function mapping

Several functions have different names in SpatiaLite than in PostGIS. The GeoAlchemy 2 package is based on the PostGIS syntax but it is possible to automatically translate the queries into SpatiaLite ones. For example, the function *ST\_GeomFromEWKT* is automatically translated into *GeomFromEWKT*. Unfortunately, only a few functions are automatically mapped (the ones internally used by GeoAlchemy 2). Nevertheless, it is possible to define new mappings in order to translate the queries automatically. Here is an example to register a mapping for the *ST\_Buffer* function:

```
>>> geoalchemy2.functions.register_sqlite_mapping(  
...     {'ST_Buffer': 'Buffer'}  
... )
```

After this command, all *ST\_Buffer* calls in the queries will be translated to *Buffer* calls when the query is executed on a SQLite DB.

A more complex example is provided for when the *PostGIS* function should be mapped depending on the given parameters. For example, the *ST\_Buffer* function can actually be translated into either the *Buffer* function or the *SingleSidedBuffer* function (only when *side=right* or *side=left* is passed). See the [Function translation for specific dialect](#) example in the gallery.

### 4.3.8 Further Reference

- GeoAlchemy 2 ORM Tutorial: [ORM Tutorial](#)
- GeoAlchemy 2 Spatial Functions Reference: [Spatial Functions](#)
- GeoAlchemy 2 Spatial Operators Reference: [Spatial Operators](#)
- GeoAlchemy 2 Elements Reference: [Elements](#)
- SpatiaLite 4.3.0 SQL functions reference list



## GALLERY

## 5.1 Gallery

### 5.1.1 Automatically use a function at insert or select

Sometimes the application wants to apply a function in an insert or in a select. For example, the application might need the geometry with lat/lon coordinates while they are projected in the DB. To avoid having to always tweak the query with a `ST_Transform()`, it is possible to define a `TypeDecorator`

```
11 import re
12
13 from sqlalchemy import Column
14 from sqlalchemy import Integer
15 from sqlalchemy import MetaData
16 from sqlalchemy import func
17 from sqlalchemy import text
18 from sqlalchemy.orm import declarative_base
19 from sqlalchemy.types import TypeDecorator
20
21 from geoalchemy2 import Geometry
22 from geoalchemy2 import shape
23
24 # Tests imports
25 from tests import test_only_with_dialects
26
27 metadata = MetaData()
28
29 Base = declarative_base(metadata=metadata)
30
31
32 class TransformedGeometry(TypeDecorator):
33     """This class is used to insert a ST_Transform() in each insert or select."""
34
35     impl = Geometry
36
37     cache_ok = True
38
39     def __init__(self, db_srid, app_srid, **kwargs):
40         kwargs["srid"] = db_srid
41         super().__init__(**kwargs)
```

(continues on next page)

(continued from previous page)

```

42     self.app_srid = app_srid
43     self.db_srid = db_srid
44
45     def column_expression(self, col):
46         """The column_expression() method is overridden to set the correct type.
47
48         This is needed so that the returned element will also be decorated. In this_
↪case we don't
49         want to transform it again afterwards so we set the same SRID to both the ``db_
↪srid`` and
50         ``app_srid`` arguments.
51         Without this the SRID of the WKBElement would be wrong.
52         """
53         return getattr(func, self.impl.as_binary)(
54             func.ST_Transform(col, self.app_srid),
55             type_=self.__class__(db_srid=self.app_srid, app_srid=self.app_srid),
56         )
57
58     def bind_expression(self, bindvalue):
59         return func.ST_Transform(
60             self.impl.bind_expression(bindvalue),
61             self.db_srid,
62             type_=self,
63         )
64
65
66 class ThreeDGeometry(TypeDecorator):
67     """This class is used to insert a ST_Force3D() in each insert."""
68
69     impl = Geometry
70
71     cache_ok = True
72
73     def column_expression(self, col):
74         """The column_expression() method is overridden to set the correct type.
75
76         This is not needed in this example but it is needed if one wants to override_
↪other methods
77         of the TypeDecorator class, like ``process_result_value()`` for example.
78         """
79         return getattr(func, self.impl.as_binary)(col, type_=self)
80
81     def bind_expression(self, bindvalue):
82         return func.ST_Force3D(
83             self.impl.bind_expression(bindvalue),
84             type_=self,
85         )
86
87
88 class Point(Base):
89     __tablename__ = "point"
90     id = Column(Integer, primary_key=True)

```

(continues on next page)



(continued from previous page)

```

91     raw_geom = Column(Geometry(srid=4326, geometry_type="POINT"))
92     geom = Column(TransformedGeometry(db_srid=2154, app_srid=4326, geometry_type="POINT
↳"))
93     three_d_geom = Column(ThreeDGeometry(srid=4326, geometry_type="POINTZ",
↳dimension=3))
94
95
96     def check_wkb(wkb, x, y):
97         pt = shape.to_shape(wkb)
98         assert round(pt.x, 5) == x
99         assert round(pt.y, 5) == y
100
101
102     @test_only_with_dialects("postgresql")
103     class TestTypeDecorator:
104         def _create_one_point(self, session, conn):
105             metadata.drop_all(conn, checkfirst=True)
106             metadata.create_all(conn)
107
108             # Create new point instance
109             p = Point()
110             p.raw_geom = "SRID=4326;POINT(5 45)"
111             p.geom = "SRID=4326;POINT(5 45)"
112             p.three_d_geom = "SRID=4326;POINT(5 45)" # Insert 2D geometry into 3D column
113
114             # Insert point
115             session.add(p)
116             session.flush()
117             session.expire(p)
118
119             return p.id
120
121         def test_transform(self, session, conn):
122             self._create_one_point(session, conn)
123
124             # Query the point and check the result
125             pt = session.query(Point).one()
126             assert pt.id == 1
127             assert pt.raw_geom.srid == 4326
128             check_wkb(pt.raw_geom, 5, 45)
129
130             assert pt.geom.srid == 4326
131             check_wkb(pt.geom, 5, 45)
132
133             # Check that the data is correct in DB using raw query
134             q = text("SELECT id, ST_AsEWKT(geom) AS geom FROM point;")
135             res_q = session.execute(q).fetchone()
136             assert res_q.id == 1
137             assert re.match(r"SRID=2154;POINT\((857581\.8993196681? 6435414\.7478354\)\"", res_
↳q.geom)
138
139             # Compare geom, raw_geom with auto transform and explicit transform

```

(continues on next page)

(continued from previous page)

```

140     pt_trans = session.query(
141         Point,
142         Point.raw_geom,
143         func.ST_Transform(Point.raw_geom, 2154).label("trans"),
144     ).one()
145
146     assert pt_trans[0].id == 1
147
148     assert pt_trans[0].geom.srid == 4326
149     check_wkb(pt_trans[0].geom, 5, 45)
150
151     assert pt_trans[0].raw_geom.srid == 4326
152     check_wkb(pt_trans[0].raw_geom, 5, 45)
153
154     assert pt_trans[1].srid == 4326
155     check_wkb(pt_trans[1], 5, 45)
156
157     assert pt_trans[2].srid == 2154
158     check_wkb(pt_trans[2], 857581.89932, 6435414.74784)
159
160     def test_force_3d(self, session, conn):
161         self._create_one_point(session, conn)
162
163         # Query the point and check the result
164         pt = session.query(Point).one()
165
166         assert pt.id == 1
167         assert pt.three_d_geom.srid == 4326
168         assert pt.three_d_geom.desc.lower() == (
169             "01010000a0e610000000000000000000144000000000008046400000000000000000"
170         )

```

## 5.1.2 Compute length on insert

It is possible to insert a geometry and ask PostgreSQL to compute its length at the same time. This example uses SQLAlchemy core queries.

```

9     from sqlalchemy import Column
10    from sqlalchemy import Float
11    from sqlalchemy import Integer
12    from sqlalchemy import MetaData
13    from sqlalchemy import Table
14    from sqlalchemy import bindparam
15    from sqlalchemy import func
16
17    from geoalchemy2 import Geometry
18    from geoalchemy2.shape import to_shape
19
20    # Tests imports
21    from tests import select
22

```

(continues on next page)

(continued from previous page)

```

23 metadata = MetaData()
24
25 table = Table(
26     "inserts",
27     metadata,
28     Column("id", Integer, primary_key=True),
29     Column("geom", Geometry("LINESTRING", 4326)),
30     Column("distance", Float),
31 )
32
33
34 class TestLengthAtInsert:
35     def test_query(self, conn):
36         metadata.drop_all(conn, checkfirst=True)
37         metadata.create_all(conn)
38
39         # Define geometries to insert
40         values = [
41             {"ewkt": "SRID=4326;LINESTRING(0 0, 1 0)"},
42             {"ewkt": "SRID=4326;LINESTRING(0 0, 0 1)"},
43         ]
44
45         # Define the query to compute distance (without spheroid)
46         distance = func.ST_Length(func.ST_GeomFromEWKT(bindparam("ewkt")), False)
47
48         i = table.insert()
49         i = i.values(geom=bindparam("ewkt"), distance=distance)
50
51         # Execute the query with values as parameters
52         conn.execute(i, values)
53
54         # Check the result
55         q = select([table])
56         res = conn.execute(q).fetchall()
57
58         # Check results
59         assert len(res) == 2
60
61         r1 = res[0]
62         assert r1[0] == 1
63         assert r1[1].srid == 4326
64         assert to_shape(r1[1]).wkt == "LINESTRING (0 0, 1 0)"
65         assert round(r1[2]) == 111195
66
67         r2 = res[1]
68         assert r2[0] == 2
69         assert r2[1].srid == 4326
70         assert to_shape(r2[1]).wkt == "LINESTRING (0 0, 0 1)"
71         assert round(r2[2]) == 111195

```

### 5.1.3 Decipher Raster

The *RasterElement* objects store the Raster data in WKB form. When using rasters it is usually better to convert them into TIFF, PNG, JPEG or whatever. Nevertheless, it is possible to decipher the WKB to get a 2D list of values. This example uses SQLAlchemy ORM queries.

```

10 import binascii
11 import struct
12
13 import pytest
14 from sqlalchemy import Column
15 from sqlalchemy import Integer
16 from sqlalchemy import MetaData
17 from sqlalchemy.orm import declarative_base
18
19 from geoalchemy2 import Raster
20 from geoalchemy2 import WKTElement
21
22 # Tests imports
23 from tests import test_only_with_dialects
24
25 metadata = MetaData()
26 Base = declarative_base(metadata=metadata)
27
28
29 class Ocean(Base):
30     __tablename__ = "ocean"
31     id = Column(Integer, primary_key=True)
32     rast = Column(Raster)
33
34     def __init__(self, rast):
35         self.rast = rast
36
37
38 def _format_e(endianness, struct_format):
39     return _ENDIANNESS[endianness] + struct_format
40
41
42 def wkbHeader(raw):
43     # Function to decipher the WKB header
44     # See http://trac.osgeo.org/postgis/browser/trunk/raster/doc/RFC2-WellKnownBinaryFormat
45
46     header = {}
47
48     header["endianness"] = struct.unpack("b", raw[0:1])[0]
49
50     e = header["endianness"]
51     header["version"] = struct.unpack(_format_e(e, "H"), raw[1:3])[0]
52     header["nbands"] = struct.unpack(_format_e(e, "H"), raw[3:5])[0]
53     header["scaleX"] = struct.unpack(_format_e(e, "d"), raw[5:13])[0]
54     header["scaleY"] = struct.unpack(_format_e(e, "d"), raw[13:21])[0]
55     header["ipX"] = struct.unpack(_format_e(e, "d"), raw[21:29])[0]

```

(continues on next page)

(continued from previous page)

```

56 header["ipY"] = struct.unpack(_format_e(e, "d"), raw[29:37])[0]
57 header["skewX"] = struct.unpack(_format_e(e, "d"), raw[37:45])[0]
58 header["skewY"] = struct.unpack(_format_e(e, "d"), raw[45:53])[0]
59 header["srid"] = struct.unpack(_format_e(e, "i"), raw[53:57])[0]
60 header["width"] = struct.unpack(_format_e(e, "H"), raw[57:59])[0]
61 header["height"] = struct.unpack(_format_e(e, "H"), raw[59:61])[0]
62
63 return header
64
65
66 def read_band(data, offset, pixtype, height, width, endianness=1):
67     ptype, _, psize = _PTYPE[pixtype]
68     pix_data = data[offset + 1 : offset + 1 + width * height * psize]
69     band = [
70         [
71             struct.unpack(
72                 _format_e(endianness, ptype),
73                 pix_data[(i * width + j) * psize : (i * width + j + 1) * psize],
74             )[0]
75             for j in range(width)
76         ]
77         for i in range(height)
78     ]
79     return band
80
81
82 def read_band_numpy(data, offset, pixtype, height, width, endianness=1):
83     import numpy as np # noqa
84
85     _, dtype, psize = _PTYPE[pixtype]
86     dt = np.dtype(dtype)
87     dt = dt.newbyteorder(_ENDIANNESS[endianness])
88     band = np.frombuffer(data, dtype=dtype, count=height * width, offset=offset + 1)
89     band = np.reshape(band, ((height, width)))
90     return band
91
92
93 _PTYPE = {
94     0: ["?", "?", 1],
95     1: ["B", "B", 1],
96     2: ["B", "B", 1],
97     3: ["b", "b", 1],
98     4: ["B", "B", 1],
99     5: ["h", "i2", 2],
100    6: ["H", "u2", 2],
101    7: ["i", "i4", 4],
102    8: ["I", "u4", 4],
103   10: ["f", "f4", 4],
104   11: ["d", "f8", 8],
105 }
106
107 _ENDIANNESS = {

```

(continues on next page)

(continued from previous page)

```

108     0: ">",
109     1: "<",
110 }
111
112
113 def wkbImage(raster_data, use_numpy=False):
114     """Function to decipher the WKB raster data"""
115
116     # Get binary data
117     raw = binascii.unhexlify(raster_data)
118
119     # Read header
120     h = wkbHeader(bytes(raw))
121     e = h["endianness"]
122
123     img = [] # array to store image bands
124     offset = 61 # header raw length in bytes
125     band_size = h["width"] * h["height"] # number of pixels in each band
126
127     for i in range(h["nbands"]):
128         # Determine pixtype for this band
129         pixtype = struct.unpack(_format_e(e, "b"), raw[offset : offset + 1])[0] - 64
130
131         # Read data with either pure Python or Numpy
132         if use_numpy:
133             band = read_band_numpy(raw, offset, pixtype, h["height"], h["width"])
134         else:
135             band = read_band(raw, offset, pixtype, h["height"], h["width"])
136
137         # Store the result
138         img.append(band)
139         offset = offset + 2 + band_size
140
141     return img
142
143
144 @test_only_with_dialects("postgres")
145 class TestDecipherRaster:
146     @pytest.mark.parametrize(
147         "pixel_type",
148         [
149             "1BB",
150             "2BUI",
151             "4BUI",
152             "8BSI",
153             "8BUI",
154             "16BSI",
155             "16BUI",
156             "32BSI",
157             "32BUI",
158             "32BF",
159             "64BF",

```

(continues on next page)

(continued from previous page)

```

160     ],
161 )
162 def test_decipher_raster(self, pixel_type, session, conn):
163     """Create a raster and decipher it"""
164     metadata.drop_all(conn, checkfirst=True)
165     metadata.create_all(conn)
166
167     # Create a new raster
168     polygon = WKTElement("POLYGON((0 0,1 1,0 1,0 0))", srid=4326)
169     o = Ocean(polygon.ST_AsRaster(5, 6, pixel_type))
170     session.add(o)
171     session.flush()
172
173     # Decipher data from each raster
174     image = wkbImage(o.rast.data)
175
176     # Define expected result
177     expected = [
178         [0, 1, 1, 1, 1],
179         [1, 1, 1, 1, 1],
180         [0, 1, 1, 1, 0],
181         [0, 1, 1, 0, 0],
182         [0, 1, 0, 0, 0],
183         [0, 0, 0, 0, 0],
184     ]
185
186     # Check results
187     band = image[0]
188     assert band == expected

```

### 5.1.4 Disable wrapping in select

If the application wants to build queries with GeoAlchemy 2 and gets them as strings, the wrapping of geometry columns with a `ST_AsEWKB()` function might be annoying. In this case it is possible to disable this wrapping. This example uses SQLAlchemy ORM queries.

```

10 from sqlalchemy import Column
11 from sqlalchemy import Integer
12 from sqlalchemy import func
13 from sqlalchemy.orm import declarative_base
14
15 from geoalchemy2 import Geometry
16
17 # Tests imports
18 from tests import select
19
20 Base = declarative_base()
21
22
23 class RawGeometry(Geometry):
24     """This class is used to remove the 'ST_AsEWKB()' function from select queries"""

```

(continues on next page)

(continued from previous page)

```

25
26     def column_expression(self, col):
27         return col
28
29
30     class Point(Base):
31         __tablename__ = "point"
32         id = Column(Integer, primary_key=True)
33         geom = Column(Geometry(srid=4326, geometry_type="POINT"))
34         raw_geom = Column(RawGeometry(srid=4326, geometry_type="POINT"))
35
36
37     def test_no_wrapping():
38         # Select all columns
39         select_query = select([Point])
40
41         # Check that the 'geom' column is wrapped by 'ST_AsEWKB()' and that the column
42         # 'raw_geom' is not.
43         assert str(select_query) == (
44             "SELECT point.id, ST_AsEWKB(point.geom) AS geom, point.raw_geom \nFROM point"
45         )
46
47
48     def test_func_no_wrapping():
49         # Select query with function
50         select_query = select(
51             [
52                 func.ST_Buffer(Point.geom), # with wrapping (default behavior)
53                 func.ST_Buffer(Point.geom, type_=Geometry), # with wrapping
54                 func.ST_Buffer(Point.geom, type_=RawGeometry), # without wrapping
55             ]
56         )
57
58         # Check the query
59         assert str(select_query) == (
60             "SELECT "
61             'ST_AsEWKB(ST_Buffer(point.geom)) AS "ST_Buffer_1", '
62             'ST_AsEWKB(ST_Buffer(point.geom)) AS "ST_Buffer_2", '
63             'ST_Buffer(point.geom) AS "ST_Buffer_3" \n'
64             "FROM point"
65         )

```



### 5.1.5 Function translation for specific dialect

Some functions have different names depending on the dialect. But sometimes one function in one dialect can be mapped to several other functions in another dialect, depending on the arguments passed. For example, the *ST\_Buffer* function in PostgreSQL can translate into 2 functions in SQLite:

1. if the buffer is two-sided (symmetric), the PostgreSQL function:

```
ST_Buffer(the_table.geom, 10)
```

should become in SQLite:

```
Buffer(the_table.geom, 10)
```

2. if the buffer is one-sided, the PostgreSQL function:

```
ST_Buffer(the_table.geom, 10, 'side=right')
```

should become in SQLite:

```
SingleSidedBuffer(the_table.geom, 10, 0)
```

This case is much more complicated than just mapping a function name and we show here how to deal with it.

This example uses SQLAlchemy core queries.

```

31 from sqlalchemy import MetaData
32 from sqlalchemy import func
33 from sqlalchemy.ext.compiler import compiles
34 from sqlalchemy.orm import declarative_base
35 from sqlalchemy.sql.expression import BindParameter
36
37 from geoalchemy2 import WKTElement
38 from geoalchemy2 import functions
39
40 # Tests imports
41 from tests import format_wkt
42 from tests import select
43
44 metadata = MetaData()
45 Base = declarative_base(metadata=metadata)
46
47
48 def _compile_buffer_default(element, compiler, **kw):
49     """Compile the element in the default case (no specific dialect).
50
51     This function should not be needed for SQLAlchemy >= 1.1.
52     """
53     return "{}({})".format("ST_Buffer", compiler.process(element.clauses, **kw))
54
55
56 def _compile_buffer_sqlite(element, compiler, **kw):
57     """Compile the element for the SQLite dialect."""
58     # Get the side parameters
59     compiled = compiler.process(element.clauses, **kw)

```

(continues on next page)

(continued from previous page)

```

60     side_params = [
61         i for i in element.clauses if isinstance(i, BindParameter) and "side" in str(i.
↪value)
62     ]
63
64     if side_params:
65         side_param = side_params[0]
66         if "right" in side_param.value:
67             # If the given side is 'right', we translate the value into 0 and switch to_
↪the sided
68             # function
69             side_param.value = 0
70             element.identifier = "SingleSidedBuffer"
71         elif "left" in side_param.value:
72             # If the given side is 'left', we translate the value into 1 and switch to_
↪the sided
73             # function
74             side_param.value = 1
75             element.identifier = "SingleSidedBuffer"
76
77     if element.identifier == "ST_Buffer":
78         # If the identifier is still the default ST_Buffer we switch to the SpatialLite_
↪function
79         element.identifier = "Buffer"
80
81     # If there is no side parameter or if the side value is 'both', we use the default_
↪function
82     return "{}({})".format(element.identifier, compiled)
83
84
85     # Register the specific compilation rules
86     compiles(functions.ST_Buffer)(_compile_buffer_default)
87     compiles(functions.ST_Buffer, "sqlite")(_compile_buffer_sqlite)
88
89
90     def test_specific_compilation(conn):
91         # Build a query with a sided buffer
92         query = select(
93             [
94                 func.ST_AsText(
95                     func.ST_Buffer(WKTElement("LINESTRING(0 0, 1 0)", srid=4326), 1,
↪"side=left")
96             )
97         ]
98     )
99
100     # Check the compiled query: the sided buffer should appear only in the SQLite query
101     compiled_query = str(query.compile(dialect=conn.dialect))
102     if conn.dialect.name == "sqlite":
103         assert "SingleSidedBuffer" in compiled_query
104         assert "ST_Buffer" not in compiled_query
105     else:

```

(continues on next page)

(continued from previous page)

```

106     assert "SingleSidedBuffer" not in compiled_query
107     assert "ST_Buffer" in compiled_query
108
109     # Check the actual result of the query
110     res = conn.execute(query).scalar()
111     assert format_wkt(res) == "POLYGON((1 0,0 0,0 1,1 1,1 0))"
112
113     # Build a query with symmetric buffer to check nothing was broken
114     query = select(
115         [func.ST_AsText(func.ST_Buffer(WKTElement("LINESTRING(0 0, 1 0)", srid=4326),
116         ↪1))])
117     )
118
119     # Check the compiled query: the sided buffer should never appear in the query
120     compiled_query = str(query.compile(dialect=conn.dialect))
121     assert "SingleSidedBuffer" not in compiled_query
122     if conn.dialect.name == "sqlite":
123         assert "ST_Buffer" not in compiled_query
124         assert "Buffer" in compiled_query
125     else:
126         assert "ST_Buffer" in compiled_query
127
128     # Check the actual result of the query
129     res = conn.execute(query).scalar()
130     assert format_wkt(res) != "POLYGON((1 0,0 0,0 1,1 1,1 0))"
131     assert format_wkt(res).startswith("POLYGON((1 1,1")

```

## 5.1.6 Reproject a Raster using ST\_Transform

The *ST\_Transform()* function (and a few others like *ST\_SnapToGrid()*) can be used on both *Geometry* and *Raster* types. In *GeoAlchemy2*, this function is only defined for *Geometry* as it can not be defined for several types at the same time. Thus using this function on *Raster* requires minor tweaking.

This example uses both SQLAlchemy core and ORM queries.

```

12 from sqlalchemy import Column
13 from sqlalchemy import Integer
14 from sqlalchemy import MetaData
15 from sqlalchemy import Table
16 from sqlalchemy import func
17 from sqlalchemy.orm import Query
18 from sqlalchemy.orm import declarative_base
19
20 from geoalchemy2 import Geometry
21 from geoalchemy2 import Raster
22
23 # Tests imports
24 from tests import select
25
26 metadata = MetaData()
27 Base = declarative_base(metadata=metadata)

```

(continues on next page)

(continued from previous page)

```

28
29 table = Table(
30     "raster_table",
31     metadata,
32     Column("id", Integer, primary_key=True),
33     Column("geom", Geometry("POLYGON", 4326)),
34     Column("rast", Raster()),
35 )
36
37
38 class RasterTable(Base):
39     __tablename__ = "raster_table_orm"
40     id = Column(Integer, primary_key=True)
41     geom = Column(Geometry("POLYGON", 4326))
42     rast = Column(Raster())
43
44     def __init__(self, rast):
45         self.rast = rast
46
47
48 def test_transform_core():
49     # Define the transform query for both the geometry and the raster in a naive way
50     wrong_query = select(
51         [func.ST_Transform(table.c.geom, 2154), func.ST_Transform(table.c.rast, 2154)]
52     )
53
54     # Check the query
55     assert str(wrong_query) == (
56         "SELECT "
57         "ST_AseWKB("
58         'ST_Transform(raster_table.geom, :ST_Transform_2)) AS "ST_Transform_1", '
59         "ST_AseWKB(" # <= Note that the raster is processed as a Geometry here
60         'ST_Transform(raster_table.rast, :ST_Transform_4)) AS "ST_Transform_3" \n'
61         "FROM raster_table"
62     )
63
64     # Define the transform query for both the geometry and the raster in the correct way
65     correct_query = select(
66         [
67             func.ST_Transform(table.c.geom, 2154),
68             func.ST_Transform(table.c.rast, 2154, type_=Raster),
69         ]
70     )
71
72     # Check the query
73     assert str(correct_query) == (
74         "SELECT "
75         "ST_AseWKB("
76         'ST_Transform(raster_table.geom, :ST_Transform_2)) AS "ST_Transform_1", '
77         "raster(" # <= This time the raster is correctly processed as a Raster
78         'ST_Transform(raster_table.rast, :ST_Transform_4)) AS "ST_Transform_3" \n'
79         "FROM raster_table"

```

(continues on next page)

(continued from previous page)

```

80     )
81
82
83     def test_transform ORM():
84         # Define the transform query for both the geometry and the raster in a naive way
85         wrong_query = Query([RasterTable.geom.ST_Transform(2154), RasterTable.rast.ST_
86         ↪ Transform(2154)])
87
88         # Check the query
89         assert str(wrong_query) == (
90             "SELECT "
91             "ST_AsEWKB("
92             'ST_Transform(raster_table_orm.geom, :ST_Transform_2)) AS "ST_Transform_1", '
93             "ST_AsEWKB(" # <= Note that the raster is processed as a Geometry here
94             'ST_Transform(raster_table_orm.rast, :ST_Transform_4)) AS "ST_Transform_3" \n'
95             "FROM raster_table_orm"
96         )
97
98         # Define the transform query for both the geometry and the raster in the correct way
99         correct_query = Query(
100             [
101                 RasterTable.geom.ST_Transform(2154),
102                 RasterTable.rast.ST_Transform(2154, type_=Raster),
103             ]
104         )
105
106         # Check the query
107         assert str(correct_query) == (
108             "SELECT "
109             "ST_AsEWKB("
110             'ST_Transform(raster_table_orm.geom, :ST_Transform_2)) AS "ST_Transform_1", '
111             "raster(" # <= This time the raster is correctly processed as a Raster
112             'ST_Transform(raster_table_orm.rast, :ST_Transform_4)) AS "ST_Transform_3" \n'
113             "FROM raster_table_orm"
114         )

```

## 5.1.7 Use CompositeType

Some functions return composite types. This example shows how to deal with this kind of functions.

```

8     import pytest
9     from pkg_resources import parse_version
10    from sqlalchemy import Column
11    from sqlalchemy import Float
12    from sqlalchemy import Integer
13    from sqlalchemy import MetaData
14    from sqlalchemy import __version__ as SA_VERSION
15    from sqlalchemy.orm import declarative_base
16
17    from geoalchemy2 import Raster
18    from geoalchemy2 import WKTElement

```

(continues on next page)

(continued from previous page)

```

19 from geoalchemy2.functions import GenericFunction
20 from geoalchemy2.types import CompositeType
21
22 # Tests imports
23 from tests import select
24 from tests import test_only_with_dialects
25
26
27 class SummaryStatsCustomType(CompositeType):
28     """Define the composite type returned by the function ST_SummaryStatsAgg."""
29
30     typemap = {
31         "count": Integer,
32         "sum": Float,
33         "mean": Float,
34         "stddev": Float,
35         "min": Float,
36         "max": Float,
37     }
38
39     cache_ok = True
40
41
42 class ST_SummaryStatsAgg(GenericFunction):
43     type = SummaryStatsCustomType
44     # Set a specific identifier to not override the actual ST_SummaryStatsAgg function
45     identifier = "ST_SummaryStatsAgg_custom"
46
47     inherit_cache = True
48
49
50 metadata = MetaData()
51 Base = declarative_base(metadata=metadata)
52
53
54 class Ocean(Base):
55     __tablename__ = "ocean"
56     id = Column(Integer, primary_key=True)
57     rast = Column(Raster)
58
59     def __init__(self, rast):
60         self.rast = rast
61
62
63 @test_only_with_dialects("postgresql")
64 class TestSTSummaryStatsAgg:
65     @pytest.mark.skipif(
66         parse_version(SA_VERSION) < parse_version("1.4"),
67         reason="requires SQLAlchemy>1.4",
68     )
69     def test_st_summary_stats_agg(self, session, conn):
70         metadata.drop_all(conn, checkfirst=True)

```

(continues on next page)

(continued from previous page)

```

71 metadata.create_all(conn)
72
73 # Create a new raster
74 polygon = WKTElement("POLYGON((0 0,1 1,0 1,0 0))", srid=4326)
75 o = Ocean(polygon.ST_AsRaster(5, 6))
76 session.add(o)
77 session.flush()
78
79 # Define the query to compute stats
80 stats_agg = select([Ocean.rast.ST_SummaryStatsAgg_custom(1, True, 1).label(
↪ "stats")])
81 stats_agg_alias = stats_agg.alias("stats_agg")
82
83 # Use these stats
84 query = select(
85     [
86         stats_agg_alias.c.stats.count.label("count"),
87         stats_agg_alias.c.stats.sum.label("sum"),
88         stats_agg_alias.c.stats.mean.label("mean"),
89         stats_agg_alias.c.stats.stddev.label("stddev"),
90         stats_agg_alias.c.stats.min.label("min"),
91         stats_agg_alias.c.stats.max.label("max"),
92     ]
93 )
94
95 # Check the query
96 assert str(query.compile(dialect=session.bind.dialect)) == (
97     "SELECT "
98     "(stats_agg.stats).count AS count, "
99     "(stats_agg.stats).sum AS sum, "
100     "(stats_agg.stats).mean AS mean, "
101     "(stats_agg.stats).stddev AS stddev, "
102     "(stats_agg.stats).min AS min, "
103     "(stats_agg.stats).max AS max \n"
104     "FROM ("
105     "SELECT "
106     "ST_SummaryStatsAgg("
107     "ocean.rast, "
108     "%(ST_SummaryStatsAgg_1)s, %(ST_SummaryStatsAgg_2)s, %(ST_SummaryStatsAgg_
↪ 3)s"
109     ") AS stats \n"
110     "FROM ocean) AS stats_agg"
111 )
112
113 # Execute the query
114 res = session.execute(query).fetchall()
115
116 # Check the result
117 assert res == [(15, 15.0, 1.0, 0.0, 1.0, 1.0)]

```

The [Gallery](#) page shows examples of the GeoAlchemy 2's functionalities.





## USE WITH ALEMBIC

### 6.1 Use Alembic with GeoAlchemy 2

The `Alembic` package is a lightweight database migration tool which is able to automatically detect the table column types.

#### 6.1.1 Interactions between Alembic and GeoAlchemy 2

Interactions between some features of Alembic and GeoAlchemy 2 may lead to errors in migration scripts, especially when using the `--autogenerate` feature of Alembic with the `spatial_index=True` feature of GeoAlchemy 2. In this case, the following errors occur:

1. the migration script misses the relevant imports from `geoalchemy2`.
2. the migration script will create the indexes of the spatial columns after the table is created, but these indexes are already automatically created during table creation, which will lead to an error.

For example, suppose the following table is defined:

```
class Lake(Base):
    __tablename__ = 'lake'
    id = Column(Integer, primary_key=True)
    geom = Column(
        Geometry(
            geometry_type='LINESTRING',
            srid=4326,
            spatial_index=True,
        )
    )
```

Then the command `alembic revision --autogenerate -m "Create new table"` will create the following migration script:

```
"""Create new table

Revision ID: <rev_id>
Revises: <down_rev_id>
Create Date: <date>

"""
from alembic import op
```

(continues on next page)

(continued from previous page)

```

import sqlalchemy as sa

# revision identifiers, used by Alembic.
revision = "<rev_id>"
down_revision = "<down_rev_id>"
branch_labels = None
depends_on = None

def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table(
        "lake",
        sa.Column("id", sa.Integer(), nullable=False),
        sa.Column(
            "geom",
            geoalchemy2.types.Geometry(
                geometry_type="LINESTRING",
                srid=4326,
                from_text="ST_GeomFromEWKT",
                name="geometry",
            ),
            nullable=True,
        ),
        sa.PrimaryKeyConstraint("id"),
    )
    op.create_index(
        "idx_lake_geom",
        "lake",
        ["geom"],
        unique=False,
        postgresql_using="gist",
        postgresql_ops={},
    )
    # ### end Alembic commands ###

def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_index(
        "idx_lake_geom",
        table_name="lake",
        postgresql_using="gist",
        postgresql_ops={},
    )
    op.drop_table("lake")
    # ### end Alembic commands ###

```

In this case, we have to do the following changes to make it work:

1. add the missing import from `geoalchemy2 import Geometry`.
2. remove the `create_index` statement in the `upgrade()` function.

3. remove the `drop_index` statement in the `downgrade()` function.

### 6.1.2 Helpers

In order to make the use of Alembic easier, a few helpers are provided in *geoalchemy2.alembic\_helpers*. These helpers can be used in the `env.py` file used by Alembic to auto-generate the migration scripts, like in the following example:

```
# ...
from geoalchemy2 import alembic_helpers
# ...

def run_migrations_offline():
    # ...
    context.configure(
        # ...
        include_object=alembic_helpers.include_object,
        process_revision_directives=alembic_helpers.writer,
        render_item=alembic_helpers.render_item,
    )
    # ...

def run_migrations_online():
    # ...
    context.configure(
        # ...
        include_object=alembic_helpers.include_object,
        process_revision_directives=alembic_helpers.writer,
        render_item=alembic_helpers.render_item,
    )
    # ...
```

As one can see, there are 3 specific functions to pass to the context:

1. *geoalchemy2.alembic\_helpers.include\_object()* ignores the internal tables managed by the spatial extensions (note that in some cases this function might need some customization, see the details in the doc of this function).
2. *geoalchemy2.alembic\_helpers.writer* adds specific spatial operations to Alembic.
3. *geoalchemy2.alembic\_helpers.render\_item()* automatically adds *GeoAlchemy2* imports into the migration scripts.

After running the `alembic revision --autogenerate -m <msg>` command, the migration script should be properly generated and should not need to be manually edited.

In this migration script you will notice specific spatial operation like *create\_geospatial\_table*, *drop\_geospatial\_table*, *add\_geospatial\_column*, *drop\_geospatial\_column*, etc. These operations can of course be edited manually in the migration scripts if you don't want to use auto-generation. All specific operations can be found in *geoalchemy2.alembic\_helpers*.

### 6.1.3 Dealing with custom types

With SQLAlchemy, users are able to define custom types, as shown in *Automatically use a function at insert or select*. In this case, you can refer to the [dedicated page of Alembic's documentation](#) for the details.

A simple solution for this case is to create a new `render_item` function to add specific imports for these custom types. For example, if your custom type is called `TheCustomType` and is defined in `my_package.custom_types`, you just have to edit the `env.py` file like the following:

```
# ...
from geoalchemy2 import alembic_helpers
from my_package.custom_types import TheCustomType
# ...

def render_item(obj_type, obj, autogen_context):
    """Apply custom rendering for selected items."""
    spatial_type = alembic_helpers.render_item(obj_type, obj, autogen_context)
    if spatial_type:
        return spatial_type

    # For the custom type
    if obj_type == 'type' and isinstance(obj, TheCustomType):
        import_name = obj.__class__.__name__
        autogen_context.imports.add(f"from my_package.custom_types import {import_name}")
        return "%r" % obj

    # default rendering for other objects
    return False

def run_migrations_offline():
    # ...
    context.configure(
        # ...
        include_object=alembic_helpers.include_object,
        process_revision_directives=alembic_helpers.writer,
        render_item=render_item,
    )
    # ...

def run_migrations_online():
    # ...
    context.configure(
        # ...
        include_object=alembic_helpers.include_object,
        process_revision_directives=alembic_helpers.writer,
        render_item=render_item,
    )
    # ...
```

Then the proper imports will be automatically added in the migration scripts.

### 6.1.4 Dialects

Some dialects (like SQLite) require some specific management to alter columns or tables. In this case, other dedicated helpers are provided to handle this. For example, if one wants to add and drop columns in a SQLite database, the Spatialite extension should be loaded when the engine connects, thus the `env.py` file should look like the following:

```
from geoalchemy2 import alembic_helpers
from geoalchemy2 import load_spatialite

def run_migrations_offline():
    # ...
    context.configure(
        # ...
        include_object=alembic_helpers.include_object,
        process_revision_directives=alembic_helpers.writer,
        render_item=alembic_helpers.render_item,
    )
    # ...

def run_migrations_online():
    # ...
    if connectable.dialect.name == "sqlite":
        # Load the Spatialite extension when the engine connects to the DB
        listen(connectable, 'connect', load_spatialite)

    with connectable.connect() as connection:
        # ...
        context.configure(
            # ...
            include_object=alembic_helpers.include_object,
            process_revision_directives=alembic_helpers.writer,
            render_item=alembic_helpers.render_item,
        )
        # ...
```

The GeoAlchemy 2 package is compatible with the migration tool [Alembic](#). The [Use Alembic with GeoAlchemy 2](#) page provides more details on this topic.



## REFERENCE DOCUMENTATION

### 7.1 Types

This module defines the Column types.

The `geoalchemy2.types.Geometry`, `geoalchemy2.types.Geography`, and `geoalchemy2.types.Raster` classes are used when defining geometry, geography and raster columns/properties in models.

**class** `geoalchemy2.types.CompositeType(*args, **kwargs)`

Bases: `UserDefinedType`

A composite type used by some spatial functions.

A wrapper for `geoalchemy2.elements.CompositeElement`, that can be used as the return type in PostgreSQL functions that return composite values.

This is used as the base class of `geoalchemy2.types.GeometryDump`.

**class** `comparator_factory(expr: ColumnElement[_CT])`

Bases: `Comparator`

**typemap** = {}

Dictionary used for defining the content types and their corresponding keys. Set in subclasses.

**class** `geoalchemy2.types.Geography(geometry_type='GEOMETRY', srid=-1, dimension=2, spatial_index=True, use_N_D_index=False, management=False, use_typedmod=None, from_text=None, name=None, nullable=True, _spatial_index_reflected=None)`

Bases: `_GISType`

The Geography type.

Creating a geography column is done like this:

```
Column(Geography(geometry_type='POINT', srid=4326))
```

See `geoalchemy2.types._GISType` for the list of arguments that can be passed to the constructor.

**ElementType**

The element class to use. Used by the parent class' `result_processor` method.

alias of `WKBElement`

**as\_binary** = 'ST\_AsBinary'

The "as binary" function to use. Used by the parent class' `column_expression` method.

**cache\_ok:** `Optional[bool] = False`

Disable cache for this type.

**from\_text** = `'ST_GeogFromText'`

The FromText geography constructor. Used by the parent class' `bind_expression` method.

**name** = `'geography'`

Type name used for defining geography columns in `CREATE TABLE`.

```
class geoalchemy2.types.Geometry(geometry_type='GEOMETRY', srid=-1, dimension=2,
                                spatial_index=True, use_N_D_index=False, management=False,
                                use_typedmod=None, from_text=None, name=None, nullable=True,
                                _spatial_index_reflected=None)
```

Bases: [`\_GISType`](#)

The Geometry type.

Creating a geometry column is done like this:

```
Column(Geometry(geometry_type='POINT', srid=4326))
```

See [`geoalchemy2.types.\_GISType`](#) for the list of arguments that can be passed to the constructor.

If `srid` is set then the `WKBElement` objects resulting from queries will have that SRID, and, when constructing the `WKBElement` objects, the SRID won't be read from the data returned by the database. If `srid` is not set (meaning it's -1) then the SRID set in `WKBElement` objects will be read from the data returned by the database.

#### **ElementType**

The element class to use. Used by the parent class' `result_processor` method.

alias of [`WKBElement`](#)

**as\_binary** = `'ST_AsEWKB'`

The "as binary" function to use. Used by the parent class' `column_expression` method.

**cache\_ok:** `Optional[bool] = False`

Disable cache for this type.

**from\_text** = `'ST_GeomFromEWKT'`

The "from text" geometry constructor. Used by the parent class' `bind_expression` method.

**name** = `'geometry'`

Type name used for defining geometry columns in `CREATE TABLE`.

```
class geoalchemy2.types.GeometryDump(*args, **kwargs)
```

Bases: [`CompositeType`](#)

The return type for functions like `ST_Dump`.

The type consists in a path and a geom field. You should normally never use this class directly.

**cache\_ok:** `Optional[bool] = True`

Enable cache for this type.

**typemap** = `{'geom': <class 'geoalchemy2.types.Geometry'>, 'path': ARRAY(Integer())}`

Dictionary defining the contents of a `geometry_dump`.



```
class geoalchemy2.types.Raster(spatial_index=True, from_text=None, name=None, nullable=True)
```

Bases: [\\_GISType](#)

The Raster column type.

Creating a raster column is done like this:

```
Column(Raster)
```

This class defines the `result_processor` method, so that raster values received from the database are converted to [geoalchemy2.elements.RasterElement](#) objects.

#### Parameters

**spatial\_index** – Indicate if a spatial index should be created. Default is True.

#### ElementType

The element class to use. Used by the parent class' `result_processor` method.

alias of [RasterElement](#)

**as\_binary** = 'raster'

The “as binary” function to use. Used by the parent class' `column_expression` method.

**cache\_ok: Optional[bool] = False**

Disable cache for this type.

#### comparator\_factory

This is the way by which spatial operators and functions are defined for raster columns.

alias of [BaseComparator](#)

**from\_text** = 'raster'

The “from text” raster constructor. Used by the parent class' `bind_expression` method.

**name** = 'raster'

Type name used for defining raster columns in CREATE TABLE.

```
class geoalchemy2.types.SummaryStats(*args, **kws)
```

Bases: [CompositeType](#)

Define the composite type returned by the function ST\_SummaryStatsAgg.

**cache\_ok: Optional[bool] = True**

Enable cache for this type.

```
typemap = {'count': <class 'sqlalchemy.sql.sqltypes.Integer'>, 'max': <class
'sqlalchemy.sql.sqltypes.Float'>, 'mean': <class 'sqlalchemy.sql.sqltypes.Float'>,
'min': <class 'sqlalchemy.sql.sqltypes.Float'>, 'stddev': <class
'sqlalchemy.sql.sqltypes.Float'>, 'sum': <class 'sqlalchemy.sql.sqltypes.Float'>}
```

Dictionary used for defining the content types and their corresponding keys. Set in subclasses.

```
class geoalchemy2.types._DummyGeometry(geometry_type='GEOMETRY', srid=-1, dimension=2,
    spatial_index=True, use_N_D_index=False, management=False,
    use_typedmod=None, from_text=None, name=None,
    nullable=True, _spatial_index_reflected=None)
```

Bases: [Geometry](#)

A dummy type only used with SQLite.

```
class geoalchemy2.types._GISType(geometry_type='GEOMETRY', srid=-1, dimension=2,
                                spatial_index=True, use_N_D_index=False, management=False,
                                use_typedmod=None, from_text=None, name=None, nullable=True,
                                _spatial_index_reflected=None)
```

Bases: `UserDefinedType`

The base class for spatial types.

This class defines `bind_expression` and `column_expression` methods that wrap column expressions in `ST_GeomFromEWKT`, `ST_GeogFromText`, or `ST_AsEWKB` calls.

This class also defines `result_processor` and `bind_processor` methods. The function returned by `result_processor` converts WKB values received from the database to `geoalchemy2.elements.WKBElement` objects. The function returned by `bind_processor` converts `geoalchemy2.elements.WKTElement` objects to EWKT strings.

#### Parameters

- **geometry\_type** – The geometry type.

Possible values are:

- "GEOMETRY",
- "POINT",
- "LINESTRING",
- "POLYGON",
- "MULTIPOINT",
- "MULTILINESTRING",
- "MULTIPOLYGON",
- "GEOMETRYCOLLECTION",
- "CURVE",
- None.

The latter is actually not supported with `geoalchemy2.types.Geography`.

When set to None then no “geometry type” constraints will be attached to the geometry type declaration. Using None here is not compatible with setting `management` to True.

Default is "GEOMETRY".

- **srid** – The SRID for this column. E.g. 4326. Default is -1.
- **dimension** – The dimension of the geometry. Default is 2.

With `management` set to True, that is when `AddGeometryColumn` is used to add the geometry column, there are two constraints:

- The `geometry_type` must not end with "ZM". This is due to PostGIS' `AddGeometryColumn` failing with ZM geometry types. Instead the “simple” geometry type (e.g. POINT rather POINTZM) should be used with `dimension` set to 4.
- When the `geometry_type` ends with "Z" or "M" then `dimension` must be set to 3.

With `management` set to False (the default) `dimension` is not taken into account, and the actual dimension is fully defined with the `geometry_type`.

- **spatial\_index** – Indicate if a spatial index should be created. Default is True.

- **use\_N\_D\_index** – Use the N-D index instead of the standard 2-D index.
- **management** – Indicate if the `AddGeometryColumn` and `DropGeometryColumn` management functions should be called when adding and dropping the geometry column. Should be set to `True` for PostGIS 1.x and SQLite. Default is `False`. Note that this option has no effect for `geoalchemy2.types.Geography`.
- **use\_typedmod** – By default PostgreSQL type modifiers are used to create the geometry column. To use check constraints instead set `use_typedmod` to `False`. By default this option is not included in the call to `AddGeometryColumn`. Note that this option is only taken into account if `management` is set to `True` and is only available for PostGIS 2.x.

**as\_binary** = `None`

The name of the “as binary” function for this type. Set in subclasses.

**bind\_expression**(*bindvalue*)

Specific `bind_expression` that automatically adds a conversion function.

**bind\_processor**(*dialect*)

Specific `bind_processor` that automatically process spatial elements.

**cache\_ok**: `Optional[bool]` = `False`

Disable cache for this type.

**column\_expression**(*col*)

Specific `column_expression` that automatically adds a conversion function.

**comparator\_factory**

This is the way by which spatial operators are defined for geometry/geography columns.

alias of `Comparator`

**from\_text** = `None`

The name of “from text” function for this type. Set in subclasses.

**name** = `None`

Name used for defining the main geo type (geometry or geography) in CREATE TABLE statements. Set in subclasses.

**result\_processor**(*dialect, coltype*)

Specific `result_processor` that automatically process spatial elements.

## 7.2 Elements

**class** `geoalchemy2.elements.HasFunction`

Bases: `object`

Base class used as a marker to know if a given element has a ‘geom\_from’ function.

**class** `geoalchemy2.elements._SpatialElement`(*data, srid=-1, extended=False*)

Bases: `HasFunction`

The base class for public spatial elements.

**Parameters**

- **data** – The first argument passed to the constructor is the data wrapped by the `_SpatialElement` object being constructed.

- **srid** – An integer representing the spatial reference system. E.g. 4326. Default value is -1, which means no/unknown reference system.
- **extended** – A boolean indicating whether the extended format (EWKT or EWKB) is used. Default is False.

**class** geoalchemy2.elements.**WKTElement**(data, srid=-1, extended=False)

Bases: [`\_SpatialElement`](#)

Instances of this class wrap a WKT or EWKT value.

Usage examples:

```
wkt_element_1 = WKTElement('POINT(5 45)')
wkt_element_2 = WKTElement('POINT(5 45)', srid=4326)
wkt_element_3 = WKTElement('SRID=4326;POINT(5 45)', extended=True)
```

**property desc**

This element's description string.

**geom\_from** = 'ST\_GeomFromText'

**geom\_from\_extended\_version** = 'ST\_GeomFromEWKT'

**class** geoalchemy2.elements.**WKBElement**(data, srid=-1, extended=False)

Bases: [`\_SpatialElement`](#)

Instances of this class wrap a WKB or EWKB value.

Geometry values read from the database are converted to instances of this type. In most cases you won't need to create `WKBElement` instances yourself.

If `extended` is True and `srid` is -1 at construction time then the SRID will be read from the EWKB data.

Note: you can create `WKBElement` objects from Shapely geometries using the [`geoalchemy2.shape.from\_shape\(\)`](#) function.

**property desc**

This element's description string.

**geom\_from** = 'ST\_GeomFromWKB'

**geom\_from\_extended\_version** = 'ST\_GeomFromEWKB'

**class** geoalchemy2.elements.**RasterElement**(data)

Bases: [`\_SpatialElement`](#)

Instances of this class wrap a raster value.

Raster values read from the database are converted to instances of this type. In most cases you won't need to create `RasterElement` instances yourself.

**property desc**

This element's description string.

**class** geoalchemy2.elements.**CompositeElement**(base, field, type\_)

Bases: `FunctionElement`

Instances of this class wrap a Postgres composite type.

## 7.3 Spatial Functions

This module defines the internals to map the spatial functions to the spatial columns.

This module defines the *GenericFunction* class, which is the base for the implementation of spatial functions in GeoAlchemy. This module is also where actual spatial functions are defined. Spatial functions supported by GeoAlchemy are defined in this module. See *GenericFunction* to know how to create new spatial functions.

**Note:** By convention the names of spatial functions are prefixed by ST\_. This is to be consistent with PostGIS', which itself is based on the SQL-MM standard.

Functions created by subclassing *GenericFunction* can be called in several ways:

- By using the func object, which is the SQLAlchemy standard way of calling a function. For example, without the ORM:

```
select([func.ST_Area(lake_table.c.geom)])
```

and with the ORM:

```
Session.query(func.ST_Area(Lake.geom))
```

- By applying the function to a geometry column. For example, without the ORM:

```
select([lake_table.c.geom.ST_Area()])
```

and with the ORM:

```
Session.query(Lake.geom.ST_Area())
```

- By applying the function to a *geoalchemy2.elements.WKBElement* object (*geoalchemy2.elements.WKBElement* is the type into which GeoAlchemy converts geometry values read from the database), or to a *geoalchemy2.elements.WKTElement* object. For example, without the ORM:

```
conn.scalar(lake['geom'].ST_Area())
```

and with the ORM:

```
session.scalar(lake.geom.ST_Area())
```

**Warning:** A few functions (like *ST\_Transform()*, *ST\_Union()*, *ST\_SnapToGrid()*, ...) can be used on several spatial types (*geoalchemy2.types.Geometry*, *geoalchemy2.types.Geography* and / or *geoalchemy2.types.Raster* types). In GeoAlchemy2, these functions are only defined for the *geoalchemy2.types.Geometry* type, as it can not be defined for several types at the same time. Therefore, using these functions on *geoalchemy2.types.Geography* or *geoalchemy2.types.Raster* requires minor tweaking to enforce the type by passing the *type\_=Geography* or *type\_=Raster* argument to the function:

```
s = select([func.ST_Transform(
    lake_table.c.raster,
    2154,
    type_=Raster)
    .label('transformed_raster')])
```

### 7.3.1 Reference

**class** `geoalchemy2.functions.AddAuth(*args, **kwargs)`

Adds an authorization token to be used in the current transaction.

see <http://postgis.net/docs/AddAuth.html>

**class** `geoalchemy2.functions.AddGeometryColumn(*args, **kwargs)`

Adds a geometry column to an existing table.

see <http://postgis.net/docs/AddGeometryColumn.html>

**class** `geoalchemy2.functions.Box2D(*args, **kwargs)`

Returns a BOX2D representing the 2D extent of the geometry.

see [http://postgis.net/docs/Box2D\\_type.html](http://postgis.net/docs/Box2D_type.html)

**class** `geoalchemy2.functions.Box3D(*args, **kwargs)`

[geometry] Returns a BOX3D representing the 3D extent of the geometry. OR [raster] Returns the box 3d representation of the enclosing box of the raster.

see [http://postgis.net/docs/Box3D\\_type.html](http://postgis.net/docs/Box3D_type.html)

**class** `geoalchemy2.functions.CheckAuth(*args, **kwargs)`

Creates a trigger on a table to prevent/allow updates and deletes of rows based on authorization token.

see <http://postgis.net/docs/CheckAuth.html>

**class** `geoalchemy2.functions.DisableLongTransactions(*args, **kwargs)`

Disables long transaction support.

see <http://postgis.net/docs/DisableLongTransactions.html>

**class** `geoalchemy2.functions.DropGeometryColumn(*args, **kwargs)`

Removes a geometry column from a spatial table.

see <http://postgis.net/docs/DropGeometryColumn.html>

**class** `geoalchemy2.functions.DropGeometryTable(*args, **kwargs)`

Drops a table and all its references in geometry\_columns.

see <http://postgis.net/docs/DropGeometryTable.html>

**class** `geoalchemy2.functions.EnableLongTransactions(*args, **kwargs)`

Enables long transaction support.

see <http://postgis.net/docs/EnableLongTransactions.html>

**class** `geoalchemy2.functions.Find_SRID(*args, **kwargs)`

Returns the SRID defined for a geometry column.

see [http://postgis.net/docs/Find\\_SRID.html](http://postgis.net/docs/Find_SRID.html)

**class** `geoalchemy2.functions.GenericFunction(*args, **kwargs)`

The base class for GeoAlchemy functions.

This class inherits from `sqlalchemy.sql.functions.GenericFunction`, so functions defined by subclassing this class can be given a fixed return type. For example, functions like *ST\_Buffer* and *ST\_Envelope* have their type attributes set to *geoalchemy2.types.Geometry*.

This class allows constructs like `Lake.geom.ST_Buffer(2)`. In that case the `Function` instance is bound to an expression (`Lake.geom` here), and that expression is passed to the function when the function is actually called.

If you need to use a function that GeoAlchemy does not provide you will certainly want to subclass this class. For example, if you need the `ST_TransScale` spatial function, which isn't (currently) natively supported by GeoAlchemy, you will write this:

```
from geoalchemy2 import Geometry
from geoalchemy2.functions import GenericFunction

class ST_TransScale(GenericFunction):
    name = 'ST_TransScale'
    type = Geometry
```

**class** `geoalchemy2.functions.GeoGenericFunction(*args, **kwargs)`

**class** `geoalchemy2.functions.GeometryType(*args, **kwargs)`

Returns the type of a geometry as text.

see <http://postgis.net/docs/GeometryType.html>

**class** `geoalchemy2.functions.LockRow(*args, **kwargs)`

Sets lock/authorization for a row in a table.

see <http://postgis.net/docs/LockRow.html>

**class** `geoalchemy2.functions.Populate_Geometry_Columns(*args, **kwargs)`

Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.

see [http://postgis.net/docs/Populate\\_Geometry\\_Columns.html](http://postgis.net/docs/Populate_Geometry_Columns.html)

**class** `geoalchemy2.functions.PostGIS_AddBBox(*args, **kwargs)`

Add bounding box to the geometry.

see [http://postgis.net/docs/PostGIS\\_AddBBox.html](http://postgis.net/docs/PostGIS_AddBBox.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.PostGIS_DropBBox(*args, **kwargs)`

Drop the bounding box cache from the geometry.

see [http://postgis.net/docs/PostGIS\\_DropBBox.html](http://postgis.net/docs/PostGIS_DropBBox.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.PostGIS_Extensions_Upgrade(*args, **kwargs)`

Packages and upgrades postgis extensions (e.g. `postgis_raster`, `postgis_topology`, `postgis_sfcgal`) to latest available version.

see [http://postgis.net/docs/PostGIS\\_Extensions\\_Upgrade.html](http://postgis.net/docs/PostGIS_Extensions_Upgrade.html)

**class** `geoalchemy2.functions.PostGIS_Full_Version(*args, **kwargs)`

Reports full postgis version and build configuration infos.

see [http://postgis.net/docs/PostGIS\\_Full\\_Version.html](http://postgis.net/docs/PostGIS_Full_Version.html)

**class** geoalchemy2.functions.**PostGIS\_GEOS\_Version**(\*args, \*\*kwargs)

Returns the version number of the GEOS library.

see [http://postgis.net/docs/PostGIS\\_GEOS\\_Version.html](http://postgis.net/docs/PostGIS_GEOS_Version.html)

**class** geoalchemy2.functions.**PostGIS\_HasBBox**(\*args, \*\*kwargs)

Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

see [http://postgis.net/docs/PostGIS\\_HasBBox.html](http://postgis.net/docs/PostGIS_HasBBox.html)

**class** geoalchemy2.functions.**PostGIS\_LibXML\_Version**(\*args, \*\*kwargs)

Returns the version number of the libxml2 library.

see [http://postgis.net/docs/PostGIS\\_LibXML\\_Version.html](http://postgis.net/docs/PostGIS_LibXML_Version.html)

**class** geoalchemy2.functions.**PostGIS\_Lib\_Build\_Date**(\*args, \*\*kwargs)

Returns build date of the PostGIS library.

see [http://postgis.net/docs/PostGIS\\_Lib\\_Build\\_Date.html](http://postgis.net/docs/PostGIS_Lib_Build_Date.html)

**class** geoalchemy2.functions.**PostGIS\_Lib\_Version**(\*args, \*\*kwargs)

Returns the version number of the PostGIS library.

see [http://postgis.net/docs/PostGIS\\_Lib\\_Version.html](http://postgis.net/docs/PostGIS_Lib_Version.html)

**class** geoalchemy2.functions.**PostGIS\_Liblwgeom\_Version**(\*args, \*\*kwargs)

Returns the version number of the liblwgeom library. This should match the version of PostGIS.

see [http://postgis.net/docs/PostGIS\\_Liblwgeom\\_Version.html](http://postgis.net/docs/PostGIS_Liblwgeom_Version.html)

**class** geoalchemy2.functions.**PostGIS\_PROJ\_Version**(\*args, \*\*kwargs)

Returns the version number of the PROJ4 library.

see [http://postgis.net/docs/PostGIS\\_PROJ\\_Version.html](http://postgis.net/docs/PostGIS_PROJ_Version.html)

**class** geoalchemy2.functions.**PostGIS\_Scripts\_Build\_Date**(\*args, \*\*kwargs)

Returns build date of the PostGIS scripts.

see [http://postgis.net/docs/PostGIS\\_Scripts\\_Build\\_Date.html](http://postgis.net/docs/PostGIS_Scripts_Build_Date.html)

**class** geoalchemy2.functions.**PostGIS\_Scripts\_Installed**(\*args, \*\*kwargs)

Returns version of the postgis scripts installed in this database.

see [http://postgis.net/docs/PostGIS\\_Scripts\\_Installed.html](http://postgis.net/docs/PostGIS_Scripts_Installed.html)

**class** geoalchemy2.functions.**PostGIS\_Scripts\_Released**(\*args, \*\*kwargs)

Returns the version number of the postgis.sql script released with the installed postgis lib.

see [http://postgis.net/docs/PostGIS\\_Scripts\\_Released.html](http://postgis.net/docs/PostGIS_Scripts_Released.html)

**class** geoalchemy2.functions.**PostGIS\_Version**(\*args, \*\*kwargs)

Returns PostGIS version number and compile-time options.

see [http://postgis.net/docs/PostGIS\\_Version.html](http://postgis.net/docs/PostGIS_Version.html)

**class** geoalchemy2.functions.**PostGIS\_Wagyu\_Version**(\*args, \*\*kwargs)

Returns the version number of the internal Wagyu library.

see [http://postgis.net/docs/PostGIS\\_Wagyu\\_Version.html](http://postgis.net/docs/PostGIS_Wagyu_Version.html)



**class** `geoalchemy2.functions.ST_3DArea(*args, **kwargs)`

Computes area of 3D surface geometries. Will return 0 for solids.

see [http://postgis.net/docs/ST\\_3DArea.html](http://postgis.net/docs/ST_3DArea.html)

**class** `geoalchemy2.functions.ST_3DClosestPoint(*args, **kwargs)`

Returns the 3D point on g1 that is closest to g2. This is the first point of the 3D shortest line.

see [http://postgis.net/docs/ST\\_3DClosestPoint.html](http://postgis.net/docs/ST_3DClosestPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DDFullyWithin(*args, **kwargs)`

Returns true if all of the 3D geometries are within the specified distance of one another.

see [http://postgis.net/docs/ST\\_3DDFullyWithin.html](http://postgis.net/docs/ST_3DDFullyWithin.html)

**class** `geoalchemy2.functions.ST_3DDWithin(*args, **kwargs)`

For 3d (z) geometry type Returns true if two geometries 3d distance is within number of units.

see [http://postgis.net/docs/ST\\_3DDWithin.html](http://postgis.net/docs/ST_3DDWithin.html)

**class** `geoalchemy2.functions.ST_3DDifference(*args, **kwargs)`

Perform 3D difference

see [http://postgis.net/docs/ST\\_3DDifference.html](http://postgis.net/docs/ST_3DDifference.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DDistance(*args, **kwargs)`

Returns the 3D cartesian minimum distance (based on spatial ref) between two geometries in projected units.

see [http://postgis.net/docs/ST\\_3DDistance.html](http://postgis.net/docs/ST_3DDistance.html)

**class** `geoalchemy2.functions.ST_3DExtent(*args, **kwargs)`

an aggregate function that returns the 3D bounding box that bounds rows of geometries.

see [http://postgis.net/docs/ST\\_3DExtent.html](http://postgis.net/docs/ST_3DExtent.html)

**class** `geoalchemy2.functions.ST_3DIntersection(*args, **kwargs)`

Perform 3D intersection

see [http://postgis.net/docs/ST\\_3DIntersection.html](http://postgis.net/docs/ST_3DIntersection.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DIntersects(*args, **kwargs)`

Returns TRUE if the Geometries “spatially intersect” in 3D - only for points, linestrings, polygons, polyhedral surface (area).

see [http://postgis.net/docs/ST\\_3DIntersects.html](http://postgis.net/docs/ST_3DIntersects.html)

**class** `geoalchemy2.functions.ST_3DLength(*args, **kwargs)`

Returns the 3D length of a linear geometry.

see [http://postgis.net/docs/ST\\_3DLength.html](http://postgis.net/docs/ST_3DLength.html)

**class** `geoalchemy2.functions.ST_3DLineInterpolatePoint(*args, **kwargs)`

Returns a point interpolated along a line in 3D. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.

see [http://postgis.net/docs/ST\\_3DLineInterpolatePoint.html](http://postgis.net/docs/ST_3DLineInterpolatePoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DLongestLine(*args, **kwargs)`

Returns the 3D longest line between two geometries

see [http://postgis.net/docs/ST\\_3DLongestLine.html](http://postgis.net/docs/ST_3DLongestLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DMakeBox(*args, **kwargs)`

Creates a BOX3D defined by two 3D point geometries.

see [http://postgis.net/docs/ST\\_3DMakeBox.html](http://postgis.net/docs/ST_3DMakeBox.html)

**class** `geoalchemy2.functions.ST_3DMaxDistance(*args, **kwargs)`

Returns the 3D cartesian maximum distance (based on spatial ref) between two geometries in projected units.

see [http://postgis.net/docs/ST\\_3DMaxDistance.html](http://postgis.net/docs/ST_3DMaxDistance.html)

**class** `geoalchemy2.functions.ST_3DPerimeter(*args, **kwargs)`

Returns the 3D perimeter of a polygonal geometry.

see [http://postgis.net/docs/ST\\_3DPerimeter.html](http://postgis.net/docs/ST_3DPerimeter.html)

**class** `geoalchemy2.functions.ST_3DShortestLine(*args, **kwargs)`

Returns the 3D shortest line between two geometries

see [http://postgis.net/docs/ST\\_3DShortestLine.html](http://postgis.net/docs/ST_3DShortestLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_3DUnion(*args, **kwargs)`

Perform 3D union

see [http://postgis.net/docs/ST\\_3DUnion.html](http://postgis.net/docs/ST_3DUnion.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_AddBand(*args, **kwargs)`

Returns a raster with the new band(s) of given type added with given initial value in the given index location. If no index is specified, the band is added to the end.

see [http://postgis.net/docs/RT\\_ST\\_AddBand.html](http://postgis.net/docs/RT_ST_AddBand.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_AddMeasure(*args, **kwargs)`

Return a derived geometry with measure elements linearly interpolated between the start and end points.

see [http://postgis.net/docs/ST\\_AddMeasure.html](http://postgis.net/docs/ST_AddMeasure.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_AddPoint(*args, **kwargs)`

Add a point to a LineString.

see [http://postgis.net/docs/ST\\_AddPoint.html](http://postgis.net/docs/ST_AddPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Affine(*args, **kwargs)`

Apply a 3D affine transformation to a geometry.

see [http://postgis.net/docs/ST\\_Affine.html](http://postgis.net/docs/ST_Affine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Angle(*args, **kwargs)`

Returns the angle between 3 points, or between 2 vectors (4 points or 2 lines).

see [http://postgis.net/docs/ST\\_Angle.html](http://postgis.net/docs/ST_Angle.html)

**class** `geoalchemy2.functions.ST_ApproximateMedialAxis(*args, **kwargs)`

Compute the approximate medial axis of an areal geometry.

see [http://postgis.net/docs/ST\\_ApproximateMedialAxis.html](http://postgis.net/docs/ST_ApproximateMedialAxis.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Area(*args, **kwargs)`

Returns the area of a polygonal geometry.

see [http://postgis.net/docs/ST\\_Area.html](http://postgis.net/docs/ST_Area.html)

**class** geoalchemy2.functions.**ST\_AsBinary**(\*args, \*\*kwargs)

[geometry] Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. OR [raster] Return the Well-Known Binary (WKB) representation of the raster.

see [http://postgis.net/docs/ST\\_AsBinary.html](http://postgis.net/docs/ST_AsBinary.html)

**class** geoalchemy2.functions.**ST\_AsEWKB**(\*args, \*\*kwargs)

Return the Well-Known Binary (WKB) representation of the geometry with SRID meta data.

see [http://postgis.net/docs/ST\\_AsEWKB.html](http://postgis.net/docs/ST_AsEWKB.html)

**class** geoalchemy2.functions.**ST\_AsEWKT**(\*args, \*\*kwargs)

Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.

see [http://postgis.net/docs/ST\\_AsEWKT.html](http://postgis.net/docs/ST_AsEWKT.html)

**class** geoalchemy2.functions.**ST\_AsEncodedPolyline**(\*args, \*\*kwargs)

Returns an Encoded Polyline from a LineString geometry.

see [http://postgis.net/docs/ST\\_AsEncodedPolyline.html](http://postgis.net/docs/ST_AsEncodedPolyline.html)

**class** geoalchemy2.functions.**ST\_AsGDALRaster**(\*args, \*\*kwargs)

Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST\_GDALDrivers() to get a list of formats supported by your library.

see [http://postgis.net/docs/RT\\_ST\\_AsGDALRaster.html](http://postgis.net/docs/RT_ST_AsGDALRaster.html)

**class** geoalchemy2.functions.**ST\_AsGML**(\*args, \*\*kwargs)

Return the geometry as a GML version 2 or 3 element.

see [http://postgis.net/docs/ST\\_AsGML.html](http://postgis.net/docs/ST_AsGML.html)

**class** geoalchemy2.functions.**ST\_AsGeoJSON**(\*args, \*\*kwargs)

Return the geometry as a GeoJSON “geometry” object, or the row as a GeoJSON feature” object (PostGIS 3 only). (Cf GeoJSON specifications RFC 7946). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example). See [https://postgis.net/docs/ST\\_AsGeoJSON.html](https://postgis.net/docs/ST_AsGeoJSON.html)

**class** geoalchemy2.functions.**ST\_AsGeobuf**(\*args, \*\*kwargs)

Return a Geobuf representation of a set of rows.

see [http://postgis.net/docs/ST\\_AsGeobuf.html](http://postgis.net/docs/ST_AsGeobuf.html)

**class** geoalchemy2.functions.**ST\_AsHEXEWKB**(\*args, \*\*kwargs)

Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.

see [http://postgis.net/docs/ST\\_AsHEXEWKB.html](http://postgis.net/docs/ST_AsHEXEWKB.html)

**class** geoalchemy2.functions.**ST\_AsHexWKB**(\*args, \*\*kwargs)

Return the Well-Known Binary (WKB) in Hex representation of the raster.

see [http://postgis.net/docs/RT\\_ST\\_AsHexWKB.html](http://postgis.net/docs/RT_ST_AsHexWKB.html)

**class** geoalchemy2.functions.**ST\_AsJPEG**(\*args, \*\*kwargs)

Return the raster tile selected bands as a single Joint Photographic Exports Group (JPEG) image (byte array). If no band is specified and 1 or more than 3 bands, then only the first band is used. If only 3 bands then all 3 bands are used and mapped to RGB.

see [http://postgis.net/docs/RT\\_ST\\_AsJPEG.html](http://postgis.net/docs/RT_ST_AsJPEG.html)

**class** `geoalchemy2.functions.ST_AsKML(*args, **kwargs)`

Return the geometry as a KML element. Several variants. Default version=2, default maxdecimaldigits=15

see [http://postgis.net/docs/ST\\_AsKML.html](http://postgis.net/docs/ST_AsKML.html)

**class** `geoalchemy2.functions.ST_AsLatLonText(*args, **kwargs)`

Return the Degrees, Minutes, Seconds representation of the given point.

see [http://postgis.net/docs/ST\\_AsLatLonText.html](http://postgis.net/docs/ST_AsLatLonText.html)

**class** `geoalchemy2.functions.ST_AsMVT(*args, **kwargs)`

Aggregate function returning a Mapbox Vector Tile representation of a set of rows.

see [http://postgis.net/docs/ST\\_AsMVT.html](http://postgis.net/docs/ST_AsMVT.html)

**class** `geoalchemy2.functions.ST_AsMVTGeom(*args, **kwargs)`

Transform a geometry into the coordinate space of a Mapbox Vector Tile.

see [http://postgis.net/docs/ST\\_AsMVTGeom.html](http://postgis.net/docs/ST_AsMVTGeom.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_AsPNG(*args, **kwargs)`

Return the raster tile selected bands as a single portable network graphics (PNG) image (byte array). If 1, 3, or 4 bands in raster and no bands are specified, then all bands are used. If more 2 or more than 4 bands and no bands specified, then only band 1 is used. Bands are mapped to RGB or RGBA space.

see [http://postgis.net/docs/RT\\_ST\\_AsPNG.html](http://postgis.net/docs/RT_ST_AsPNG.html)

**class** `geoalchemy2.functions.ST_AsRaster(*args, **kwargs)`

Converts a PostGIS geometry to a PostGIS raster.

see [http://postgis.net/docs/RT\\_ST\\_AsRaster.html](http://postgis.net/docs/RT_ST_AsRaster.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_AsSVG(*args, **kwargs)`

Returns SVG path data for a geometry.

see [http://postgis.net/docs/ST\\_AsSVG.html](http://postgis.net/docs/ST_AsSVG.html)

**class** `geoalchemy2.functions.ST_AsTIFF(*args, **kwargs)`

Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

see [http://postgis.net/docs/RT\\_ST\\_AsTIFF.html](http://postgis.net/docs/RT_ST_AsTIFF.html)

**class** `geoalchemy2.functions.ST_AsTWKB(*args, **kwargs)`

Returns the geometry as TWKB, aka “Tiny Well-Known Binary”

see [http://postgis.net/docs/ST\\_AsTWKB.html](http://postgis.net/docs/ST_AsTWKB.html)

**class** `geoalchemy2.functions.ST_AsText(*args, **kwargs)`

Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

see [http://postgis.net/docs/ST\\_AsText.html](http://postgis.net/docs/ST_AsText.html)

**class** `geoalchemy2.functions.ST_AsWKB(*args, **kwargs)`

Return the Well-Known Binary (WKB) representation of the raster.

see [http://postgis.net/docs/RT\\_ST\\_AsBinary.html](http://postgis.net/docs/RT_ST_AsBinary.html)

**class** `geoalchemy2.functions.ST_AsX3D(*args, **kwargs)`

Returns a Geometry in X3D xml node element format: ISO-IEC-19776-1.2-X3DEncodings-XML

see [http://postgis.net/docs/ST\\_AsX3D.html](http://postgis.net/docs/ST_AsX3D.html)

**class** `geoalchemy2.functions.ST_Aspect(*args, **kwargs)`

Returns the aspect (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

see [http://postgis.net/docs/RT\\_ST\\_Aspect.html](http://postgis.net/docs/RT_ST_Aspect.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_Azimuth(*args, **kwargs)`

Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on pointA to pointB.

see [http://postgis.net/docs/ST\\_Azimuth.html](http://postgis.net/docs/ST_Azimuth.html)

**class** `geoalchemy2.functions.ST_Band(*args, **kwargs)`

Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters.

see [http://postgis.net/docs/RT\\_ST\\_Band.html](http://postgis.net/docs/RT_ST_Band.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_BandFileSize(*args, **kwargs)`

Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_BandFileSize.html](http://postgis.net/docs/RT_ST_BandFileSize.html)

**class** `geoalchemy2.functions.ST_BandFileTimestamp(*args, **kwargs)`

Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_BandFileTimestamp.html](http://postgis.net/docs/RT_ST_BandFileTimestamp.html)

**class** `geoalchemy2.functions.ST_BandIsNoData(*args, **kwargs)`

Returns true if the band is filled with only nodata values.

see [http://postgis.net/docs/RT\\_ST\\_BandIsNoData.html](http://postgis.net/docs/RT_ST_BandIsNoData.html)

**class** `geoalchemy2.functions.ST_BandMetaData(*args, **kwargs)`

Returns basic meta data for a specific raster band. band num 1 is assumed if none-specified.

see [http://postgis.net/docs/RT\\_ST\\_BandMetaData.html](http://postgis.net/docs/RT_ST_BandMetaData.html)

**class** `geoalchemy2.functions.ST_BandNoDataValue(*args, **kwargs)`

Returns the value in a given band that represents no data. If no band num 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_BandNoDataValue.html](http://postgis.net/docs/RT_ST_BandNoDataValue.html)

**class** `geoalchemy2.functions.ST_BandPath(*args, **kwargs)`

Returns system file path to a band stored in file system. If no bandnum specified, 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_BandPath.html](http://postgis.net/docs/RT_ST_BandPath.html)

**class** `geoalchemy2.functions.ST_BandPixelType(*args, **kwargs)`

Returns the type of pixel for given band. If no bandnum specified, 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_BandPixelType.html](http://postgis.net/docs/RT_ST_BandPixelType.html)

**class** `geoalchemy2.functions.ST_BdMPolyFromText(*args, **kwargs)`

Construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation Well-Known text representation.

see [http://postgis.net/docs/ST\\_BdMPolyFromText.html](http://postgis.net/docs/ST_BdMPolyFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_BdPolyFromText(*args, **kwargs)`

Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

see [http://postgis.net/docs/ST\\_BdPolyFromText.html](http://postgis.net/docs/ST_BdPolyFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Boundary(*args, **kwargs)`

Returns the boundary of a geometry.

see [http://postgis.net/docs/ST\\_Boundary.html](http://postgis.net/docs/ST_Boundary.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_BoundingDiagonal(*args, **kwargs)`

Returns the diagonal of a geometry's bounding box.

see [http://postgis.net/docs/ST\\_BoundingDiagonal.html](http://postgis.net/docs/ST_BoundingDiagonal.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Box2dFromGeoHash(*args, **kwargs)`

Return a BOX2D from a GeoHash string.

see [http://postgis.net/docs/ST\\_Box2dFromGeoHash.html](http://postgis.net/docs/ST_Box2dFromGeoHash.html)

**class** `geoalchemy2.functions.ST_Buffer(*args, **kwargs)`

(T) Returns a geometry covering all points within a given distance from the input geometry.

see [http://postgis.net/docs/ST\\_Buffer.html](http://postgis.net/docs/ST_Buffer.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_BuildArea(*args, **kwargs)`

Creates an areal geometry formed by the constituent linework of given geometry

see [http://postgis.net/docs/ST\\_BuildArea.html](http://postgis.net/docs/ST_BuildArea.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_CPAWithin(*args, **kwargs)`

Returns true if the closest point of approach of two trajectories is within the specified distance.

see [http://postgis.net/docs/ST\\_CPAWithin.html](http://postgis.net/docs/ST_CPAWithin.html)

**class** `geoalchemy2.functions.ST_Centroid(*args, **kwargs)`

Returns the geometric center of a geometry.

see [http://postgis.net/docs/ST\\_Centroid.html](http://postgis.net/docs/ST_Centroid.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ChaikinSmoothing(*args, **kwargs)`

Returns a “smoothed” version of the given geometry using the Chaikin algorithm

see [http://postgis.net/docs/ST\\_ChaikinSmoothing.html](http://postgis.net/docs/ST_ChaikinSmoothing.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Clip(*args, **kwargs)`

Returns the raster clipped by the input geometry. If band number not is specified, all bands are processed. If crop is not specified or TRUE, the output raster is cropped.

see [http://postgis.net/docs/RT\\_ST\\_Clip.html](http://postgis.net/docs/RT_ST_Clip.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_ClipByBox2D(*args, **kwargs)`

Returns the portion of a geometry falling within a rectangle.

see [http://postgis.net/docs/ST\\_ClipByBox2D.html](http://postgis.net/docs/ST_ClipByBox2D.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*



**class** `geoalchemy2.functions.ST_ClosestPoint(*args, **kwargs)`

Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line.

see [http://postgis.net/docs/ST\\_ClosestPoint.html](http://postgis.net/docs/ST_ClosestPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ClosestPointOfApproach(*args, **kwargs)`

Returns the measure at which points interpolated along two trajectories are closest.

see [http://postgis.net/docs/ST\\_ClosestPointOfApproach.html](http://postgis.net/docs/ST_ClosestPointOfApproach.html)

**class** `geoalchemy2.functions.ST_ClusterDBSCAN(*args, **kwargs)`

Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

see [http://postgis.net/docs/ST\\_ClusterDBSCAN.html](http://postgis.net/docs/ST_ClusterDBSCAN.html)

**class** `geoalchemy2.functions.ST_ClusterIntersecting(*args, **kwargs)`

Aggregate function that clusters the input geometries into connected sets.

see [http://postgis.net/docs/ST\\_ClusterIntersecting.html](http://postgis.net/docs/ST_ClusterIntersecting.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ClusterKMeans(*args, **kwargs)`

Window function that returns a cluster id for each input geometry using the K-means algorithm.

see [http://postgis.net/docs/ST\\_ClusterKMeans.html](http://postgis.net/docs/ST_ClusterKMeans.html)

**class** `geoalchemy2.functions.ST_ClusterWithin(*args, **kwargs)`

Aggregate function that clusters the input geometries by separation distance.

see [http://postgis.net/docs/ST\\_ClusterWithin.html](http://postgis.net/docs/ST_ClusterWithin.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Collect(*args, **kwargs)`

Creates a GeometryCollection or Multi\* geometry from a set of geometries.

see [http://postgis.net/docs/ST\\_Collect.html](http://postgis.net/docs/ST_Collect.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_CollectionExtract(*args, **kwargs)`

Given a (multi)geometry, return a (multi)geometry consisting only of elements of the specified type.

see [http://postgis.net/docs/ST\\_CollectionExtract.html](http://postgis.net/docs/ST_CollectionExtract.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_CollectionHomogenize(*args, **kwargs)`

Given a geometry collection, return the “simplest” representation of the contents.

see [http://postgis.net/docs/ST\\_CollectionHomogenize.html](http://postgis.net/docs/ST_CollectionHomogenize.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ColorMap(*args, **kwargs)`

Creates a new raster of up to four 8BUI bands (grayscale, RGB, RGBA) from the source raster and a specified band. Band 1 is assumed if not specified.

see [http://postgis.net/docs/RT\\_ST\\_ColorMap.html](http://postgis.net/docs/RT_ST_ColorMap.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_ConcaveHull(*args, **kwargs)`

The concave hull of a geometry represents a possibly concave geometry that encloses all geometries within the set. You can think of it as shrink wrapping.

see [http://postgis.net/docs/ST\\_ConcaveHull.html](http://postgis.net/docs/ST_ConcaveHull.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ConstrainedDelaunayTriangles(*args, **kwargs)`

Return a constrained Delaunay triangulation around the given input geometry.

see [http://postgis.net/docs/ST\\_ConstrainedDelaunayTriangles.html](http://postgis.net/docs/ST_ConstrainedDelaunayTriangles.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Contains(*args, **kwargs)`

[geometry] Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. OR [raster] Return true if no points of raster rastB lie in the exterior of raster rastA and at least one point of the interior of rastB lies in the interior of rastA.

see [http://postgis.net/docs/ST\\_Contains.html](http://postgis.net/docs/ST_Contains.html)

**class** `geoalchemy2.functions.ST_ContainsProperly(*args, **kwargs)`

[geometry] Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself. OR [raster] Return true if rastB intersects the interior of rastA but not the boundary or exterior of rastA.

see [http://postgis.net/docs/ST\\_ContainsProperly.html](http://postgis.net/docs/ST_ContainsProperly.html)

**class** geoalchemy2.functions.**ST\_ConvexHull**(\*args, \*\*kwargs)

[geometry] Computes the convex hull of a geometry. OR [raster] Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular shaped and non-skewed rasters, this gives the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.

see [http://postgis.net/docs/ST\\_ConvexHull.html](http://postgis.net/docs/ST_ConvexHull.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_CoordDim**(\*args, \*\*kwargs)

Return the coordinate dimension of a geometry.

see [http://postgis.net/docs/ST\\_CoordDim.html](http://postgis.net/docs/ST_CoordDim.html)

**class** geoalchemy2.functions.**ST\_Count**(\*args, \*\*kwargs)

Returns the number of pixels in a given band of a raster or raster coverage. If no band is specified defaults to band 1. If exclude\_nodata\_value is set to true, will only count pixels that are not equal to the nodata value.

see [http://postgis.net/docs/RT\\_ST\\_Count.html](http://postgis.net/docs/RT_ST_Count.html)

**class** geoalchemy2.functions.**ST\_CountAgg**(\*args, \*\*kwargs)

Aggregate. Returns the number of pixels in a given band of a set of rasters. If no band is specified defaults to band 1. If exclude\_nodata\_value is set to true, will only count pixels that are not equal to the NODATA value.

see [http://postgis.net/docs/RT\\_ST\\_CountAgg.html](http://postgis.net/docs/RT_ST_CountAgg.html)

**class** geoalchemy2.functions.**ST\_CoveredBy**(\*args, \*\*kwargs)

[geometry] Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B OR [raster] Return true if no points of raster rastA lie outside raster rastB.

see [http://postgis.net/docs/ST\\_CoveredBy.html](http://postgis.net/docs/ST_CoveredBy.html)

**class** geoalchemy2.functions.**ST\_Covers**(\*args, \*\*kwargs)

[geometry] Returns 1 (TRUE) if no point in Geometry B is outside Geometry A OR [raster] Return true if no points of raster rastB lie outside raster rastA.

see [http://postgis.net/docs/ST\\_Covers.html](http://postgis.net/docs/ST_Covers.html)

**class** geoalchemy2.functions.**ST\_Crosses**(\*args, \*\*kwargs)

Returns TRUE if the supplied geometries have some, but not all, interior points in common.

see [http://postgis.net/docs/ST\\_Crosses.html](http://postgis.net/docs/ST_Crosses.html)

**class** geoalchemy2.functions.**ST\_CurveToLine**(\*args, \*\*kwargs)

Converts a CIRCULARSTRING/CURVEPOLYGON/MULTISURFACE to a LINestring/POLYGON/MULTIPOLYGON

see [http://postgis.net/docs/ST\\_CurveToLine.html](http://postgis.net/docs/ST_CurveToLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_DFullyWithin**(\*args, \*\*kwargs)

[geometry] Returns true if all of the geometries are within the specified distance of one another OR [raster] Return true if rasters rastA and rastB are fully within the specified distance of each other.

see [http://postgis.net/docs/ST\\_DFullyWithin.html](http://postgis.net/docs/ST_DFullyWithin.html)

**class** `geoalchemy2.functions.ST_DWithin(*args, **kwargs)`

[geometry] Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and for geography units are in meters and measurement is defaulted to `use_spheroid=true` (measure around spheroid), for faster check, `use_spheroid=false` to measure along sphere. OR [raster] Return true if rasters `rastA` and `rastB` are within the specified distance of each other.

see [http://postgis.net/docs/ST\\_DWithin.html](http://postgis.net/docs/ST_DWithin.html)

**class** `geoalchemy2.functions.ST_DelaunayTriangles(*args, **kwargs)`

Return a Delaunay triangulation around the given input points.

see [http://postgis.net/docs/ST\\_DelaunayTriangles.html](http://postgis.net/docs/ST_DelaunayTriangles.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Difference(*args, **kwargs)`

Returns a geometry that represents that part of geometry A that does not intersect with geometry B.

see [http://postgis.net/docs/ST\\_Difference.html](http://postgis.net/docs/ST_Difference.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Dimension(*args, **kwargs)`

Returns the topological dimension of a geometry.

see [http://postgis.net/docs/ST\\_Dimension.html](http://postgis.net/docs/ST_Dimension.html)

**class** `geoalchemy2.functions.ST_Disjoint(*args, **kwargs)`

[geometry] Returns TRUE if the Geometries do not “spatially intersect” - if they do not share any space together. OR [raster] Return true if raster `rastA` does not spatially intersect `rastB`.

see [http://postgis.net/docs/ST\\_Disjoint.html](http://postgis.net/docs/ST_Disjoint.html)

**class** `geoalchemy2.functions.ST_Distance(*args, **kwargs)`

Returns the distance between two geometry or geography values.

see [http://postgis.net/docs/ST\\_Distance.html](http://postgis.net/docs/ST_Distance.html)

**class** `geoalchemy2.functions.ST_DistanceCPA(*args, **kwargs)`

Returns the distance between the closest point of approach of two trajectories.

see [http://postgis.net/docs/ST\\_DistanceCPA.html](http://postgis.net/docs/ST_DistanceCPA.html)

**class** `geoalchemy2.functions.ST_DistanceSphere(*args, **kwargs)`

Returns minimum distance in meters between two lon/lat geometries using a spherical earth model.

see [http://postgis.net/docs/ST\\_DistanceSphere.html](http://postgis.net/docs/ST_DistanceSphere.html)

**class** `geoalchemy2.functions.ST_DistanceSpheroid(*args, **kwargs)`

Returns the minimum distance between two lon/lat geometries using a spheroidal earth model.

see [http://postgis.net/docs/ST\\_DistanceSpheroid.html](http://postgis.net/docs/ST_DistanceSpheroid.html)

**class** `geoalchemy2.functions.ST_Distance_Sphere(*args, **kwargs)`

Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than `ST_Distance_Spheroid`, but less accurate. PostGIS versions prior to 1.5 only implemented for points.

see [http://postgis.net/docs/ST\\_Distance\\_Sphere.html](http://postgis.net/docs/ST_Distance_Sphere.html)

**class** `geoalchemy2.functions.ST_Distinct4ma(*args, **kwargs)`

Raster processing function that calculates the number of unique pixel values in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Distinct4ma.html](http://postgis.net/docs/RT_ST_Distinct4ma.html)

**class** `geoalchemy2.functions.ST_Dump(*args, **kwargs)`

Returns a set of geometry\_dump rows for the components of a geometry.

see [http://postgis.net/docs/ST\\_Dump.html](http://postgis.net/docs/ST_Dump.html)

Return type: `geoalchemy2.types.GeometryDump`.

**type**

alias of `GeometryDump`

**class** `geoalchemy2.functions.ST_DumpAsPolygons(*args, **kwargs)`

Returns a set of geomval (geom,val) rows, from a given raster band. If no band number is specified, band number defaults to 1.

see [http://postgis.net/docs/RT\\_ST\\_DumpAsPolygons.html](http://postgis.net/docs/RT_ST_DumpAsPolygons.html)

**class** `geoalchemy2.functions.ST_DumpPoints(*args, **kwargs)`

Returns a set of geometry\_dump rows for the points in a geometry.

see [http://postgis.net/docs/ST\\_DumpPoints.html](http://postgis.net/docs/ST_DumpPoints.html)

Return type: `geoalchemy2.types.GeometryDump`.

**type**

alias of `GeometryDump`

**class** `geoalchemy2.functions.ST_DumpRings(*args, **kwargs)`

Returns a set of geometry\_dump rows for the exterior and interior rings of a Polygon.

see [http://postgis.net/docs/ST\\_DumpRings.html](http://postgis.net/docs/ST_DumpRings.html)

Return type: `geoalchemy2.types.GeometryDump`.

**type**

alias of `GeometryDump`

**class** `geoalchemy2.functions.ST_DumpValues(*args, **kwargs)`

Get the values of the specified band as a 2-dimension array.

see [http://postgis.net/docs/RT\\_ST\\_DumpValues.html](http://postgis.net/docs/RT_ST_DumpValues.html)

**class** `geoalchemy2.functions.ST_EndPoint(*args, **kwargs)`

Returns the last point of a LineString or CircularLineString.

see [http://postgis.net/docs/ST\\_EndPoint.html](http://postgis.net/docs/ST_EndPoint.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** geoalchemy2.functions.**ST\_Envelope**(\*args, \*\*kwargs)

[geometry] Returns a geometry representing the bounding box of a geometry. OR [raster] Returns the polygon representation of the extent of the raster.

see [http://postgis.net/docs/ST\\_Envelope.html](http://postgis.net/docs/ST_Envelope.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_Equals**(\*args, \*\*kwargs)

Returns true if the given geometries represent the same geometry. Directionality is ignored.

see [http://postgis.net/docs/ST\\_Equals.html](http://postgis.net/docs/ST_Equals.html)

**class** geoalchemy2.functions.**ST\_EstimatedExtent**(\*args, \*\*kwargs)

Return the 'estimated' extent of a spatial table.

see [http://postgis.net/docs/ST\\_EstimatedExtent.html](http://postgis.net/docs/ST_EstimatedExtent.html)

**class** geoalchemy2.functions.**ST\_Expand**(\*args, \*\*kwargs)

Returns a bounding box expanded from another bounding box or a geometry.

see [http://postgis.net/docs/ST\\_Expand.html](http://postgis.net/docs/ST_Expand.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_Extent**(\*args, \*\*kwargs)

an aggregate function that returns the bounding box that bounds rows of geometries.

see [http://postgis.net/docs/ST\\_Extent.html](http://postgis.net/docs/ST_Extent.html)

**class** geoalchemy2.functions.**ST\_ExteriorRing**(\*args, \*\*kwargs)

Returns a LineString representing the exterior ring of a Polygon.

see [http://postgis.net/docs/ST\\_ExteriorRing.html](http://postgis.net/docs/ST_ExteriorRing.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_Extrude**(\*args, \*\*kwargs)

Extrude a surface to a related volume

see [http://postgis.net/docs/ST\\_Extrude.html](http://postgis.net/docs/ST_Extrude.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_FilterByM**(\*args, \*\*kwargs)

Filters vertex points based on their m-value

see [http://postgis.net/docs/ST\\_FilterByM.html](http://postgis.net/docs/ST_FilterByM.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_FlipCoordinates(*args, **kwargs)`

Returns a version of the given geometry with X and Y axis flipped. Useful for people who have built latitude/longitude features and need to fix them.

see [http://postgis.net/docs/ST\\_FlipCoordinates.html](http://postgis.net/docs/ST_FlipCoordinates.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Force2D(*args, **kwargs)`

Force the geometries into a “2-dimensional mode”.

see [http://postgis.net/docs/ST\\_Force2D.html](http://postgis.net/docs/ST_Force2D.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Force3D(*args, **kwargs)`

Force the geometries into XYZ mode. This is an alias for ST\_Force3DZ.

see [http://postgis.net/docs/ST\\_Force\\_3D.html](http://postgis.net/docs/ST_Force_3D.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Force3DM(*args, **kwargs)`

Force the geometries into XYM mode.

see [http://postgis.net/docs/ST\\_Force\\_3DZ.html](http://postgis.net/docs/ST_Force_3DZ.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Force3DZ(*args, **kwargs)`

Force the geometries into XYZ mode.

see [http://postgis.net/docs/ST\\_Force\\_3DZ.html](http://postgis.net/docs/ST_Force_3DZ.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Force4D(*args, **kwargs)`

Force the geometries into XYZM mode.

see [http://postgis.net/docs/ST\\_Force\\_4D.html](http://postgis.net/docs/ST_Force_4D.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForceCollection(*args, **kwargs)`

Convert the geometry into a GEOMETRYCOLLECTION.

see [http://postgis.net/docs/ST\\_Force\\_Collection.html](http://postgis.net/docs/ST_Force_Collection.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForceCurve(*args, **kwargs)`

Upcast a geometry into its curved type, if applicable.

see [http://postgis.net/docs/ST\\_ForceCurve.html](http://postgis.net/docs/ST_ForceCurve.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForceLHR(*args, **kwargs)`

Force LHR orientation

see [http://postgis.net/docs/ST\\_ForceLHR.html](http://postgis.net/docs/ST_ForceLHR.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForcePolygonCCW(*args, **kwargs)`

Orients all exterior rings counter-clockwise and all interior rings clockwise.

see [http://postgis.net/docs/ST\\_ForcePolygonCCW.html](http://postgis.net/docs/ST_ForcePolygonCCW.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForcePolygonCW(*args, **kwargs)`

Orients all exterior rings clockwise and all interior rings counter-clockwise.

see [http://postgis.net/docs/ST\\_ForcePolygonCW.html](http://postgis.net/docs/ST_ForcePolygonCW.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ForceRHR(*args, **kwargs)`

Force the orientation of the vertices in a polygon to follow the Right-Hand-Rule.

see [http://postgis.net/docs/ST\\_ForceRHR.html](http://postgis.net/docs/ST_ForceRHR.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*



**class** `geoalchemy2.functions.ST_ForceSFS(*args, **kwargs)`

Force the geometries to use SFS 1.1 geometry types only.

see [http://postgis.net/docs/ST\\_ForceSFS.html](http://postgis.net/docs/ST_ForceSFS.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_FrechetDistance(*args, **kwargs)`

Returns the Fréchet distance between two geometries.

see [http://postgis.net/docs/ST\\_FrechetDistance.html](http://postgis.net/docs/ST_FrechetDistance.html)

**class** `geoalchemy2.functions.ST_FromGDALRaster(*args, **kwargs)`

Returns a raster from a supported GDAL raster file.

see [http://postgis.net/docs/RT\\_ST\\_FromGDALRaster.html](http://postgis.net/docs/RT_ST_FromGDALRaster.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_GMLToSQL(*args, **kwargs)`

Return a specified ST\_Geometry value from GML representation. This is an alias name for ST\_GeomFromGML

see [http://postgis.net/docs/ST\\_GMLToSQL.html](http://postgis.net/docs/ST_GMLToSQL.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeneratePoints(*args, **kwargs)`

Converts a polygon or multi-polygon into a multi-point composed of randomly location points within the original areas.

see [http://postgis.net/docs/ST\\_GeneratePoints.html](http://postgis.net/docs/ST_GeneratePoints.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeoHash(*args, **kwargs)`

Return a GeoHash representation of the geometry.

see [http://postgis.net/docs/ST\\_GeoHash.html](http://postgis.net/docs/ST_GeoHash.html)

**class** `geoalchemy2.functions.ST_GeoReference(*args, **kwargs)`

Returns the georeference meta data in GDAL or ESRI format as commonly seen in a world file. Default is GDAL.

see [http://postgis.net/docs/RT\\_ST\\_GeoReference.html](http://postgis.net/docs/RT_ST_GeoReference.html)

**class** `geoalchemy2.functions.ST_GeogFromText(*args, **kwargs)`

Return a specified geography value from Well-Known Text representation or extended (WKT).

see [http://postgis.net/docs/ST\\_GeogFromText.html](http://postgis.net/docs/ST_GeogFromText.html)

Return type: *geoalchemy2.types.Geography*.

**type**

alias of *Geography*

**class** `geoalchemy2.functions.ST_GeogFromWKB(*args, **kwargs)`

Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).

see [http://postgis.net/docs/ST\\_GeogFromWKB.html](http://postgis.net/docs/ST_GeogFromWKB.html)

Return type: *geoalchemy2.types.Geography*.

**type**

alias of *Geography*

**class** `geoalchemy2.functions.ST_GeographyFromText(*args, **kwargs)`

Return a specified geography value from Well-Known Text representation or extended (WKT).

see [http://postgis.net/docs/ST\\_GeographyFromText.html](http://postgis.net/docs/ST_GeographyFromText.html)

Return type: *geoalchemy2.types.Geography*.

**type**

alias of *Geography*

**class** `geoalchemy2.functions.ST_GeomCollFromText(*args, **kwargs)`

Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

see [http://postgis.net/docs/ST\\_GeomCollFromText.html](http://postgis.net/docs/ST_GeomCollFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromEWKB(*args, **kwargs)`

Return a specified ST\_Geometry value from Extended Well-Known Binary representation (EWKB).

see [http://postgis.net/docs/ST\\_GeomFromEWKB.html](http://postgis.net/docs/ST_GeomFromEWKB.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromEWKT(*args, **kwargs)`

Return a specified ST\_Geometry value from Extended Well-Known Text representation (EWKT).

see [http://postgis.net/docs/ST\\_GeomFromEWKT.html](http://postgis.net/docs/ST_GeomFromEWKT.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromGML(*args, **kwargs)`

Takes as input GML representation of geometry and outputs a PostGIS geometry object

see [http://postgis.net/docs/ST\\_GeomFromGML.html](http://postgis.net/docs/ST_GeomFromGML.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromGeoHash(*args, **kwargs)`

Return a geometry from a GeoHash string.

see [http://postgis.net/docs/ST\\_GeomFromGeoHash.html](http://postgis.net/docs/ST_GeomFromGeoHash.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromGeoJSON(*args, **kwargs)`

Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object

see [http://postgis.net/docs/ST\\_GeomFromGeoJSON.html](http://postgis.net/docs/ST_GeomFromGeoJSON.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromKML(*args, **kwargs)`

Takes as input KML representation of geometry and outputs a PostGIS geometry object

see [http://postgis.net/docs/ST\\_GeomFromKML.html](http://postgis.net/docs/ST_GeomFromKML.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromTWKB(*args, **kwargs)`

Creates a geometry instance from a TWKB (“Tiny Well-Known Binary”) geometry representation.

see [http://postgis.net/docs/ST\\_GeomFromTWKB.html](http://postgis.net/docs/ST_GeomFromTWKB.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromText(*args, **kwargs)`

Return a specified ST\_Geometry value from Well-Known Text representation (WKT).

see [http://postgis.net/docs/ST\\_GeomFromText.html](http://postgis.net/docs/ST_GeomFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_GeomFromWKB(*args, **kwargs)`

Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.

see [http://postgis.net/docs/ST\\_GeomFromWKB.html](http://postgis.net/docs/ST_GeomFromWKB.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_GeometricMedian**(\*args, \*\*kwargs)

Returns the geometric median of a MultiPoint.

see [http://postgis.net/docs/ST\\_GeometricMedian.html](http://postgis.net/docs/ST_GeometricMedian.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_GeometryFromText**(\*args, \*\*kwargs)

Return a specified ST\_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST\_GeomFromText

see [http://postgis.net/docs/ST\\_GeometryFromText.html](http://postgis.net/docs/ST_GeometryFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_GeometryN**(\*args, \*\*kwargs)

Return the Nth geometry element of a geometry collection.

see [http://postgis.net/docs/ST\\_GeometryN.html](http://postgis.net/docs/ST_GeometryN.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_GeometryType**(\*args, \*\*kwargs)

Returns the SQL-MM type of a geometry as text.

see [http://postgis.net/docs/ST\\_GeometryType.html](http://postgis.net/docs/ST_GeometryType.html)

**class** geoalchemy2.functions.**ST\_Grayscale**(\*args, \*\*kwargs)

Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

see [http://postgis.net/docs/RT\\_ST\\_Grayscale.html](http://postgis.net/docs/RT_ST_Grayscale.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** geoalchemy2.functions.**ST\_HasArc**(\*args, \*\*kwargs)

Tests if a geometry contains a circular arc

see [http://postgis.net/docs/ST\\_HasArc.html](http://postgis.net/docs/ST_HasArc.html)

**class** geoalchemy2.functions.**ST\_HasNoBand**(\*args, \*\*kwargs)

Returns true if there is no band with given band number. If no band number is specified, then band number 1 is assumed.

see [http://postgis.net/docs/RT\\_ST\\_HasNoBand.html](http://postgis.net/docs/RT_ST_HasNoBand.html)

**class** geoalchemy2.functions.**ST\_HausdorffDistance**(\*args, \*\*kwargs)

Returns the Hausdorff distance between two geometries.

see [http://postgis.net/docs/ST\\_HausdorffDistance.html](http://postgis.net/docs/ST_HausdorffDistance.html)

**class** `geoalchemy2.functions.ST_Height(*args, **kwargs)`

Returns the height of the raster in pixels.

see [http://postgis.net/docs/RT\\_ST\\_Height.html](http://postgis.net/docs/RT_ST_Height.html)

**class** `geoalchemy2.functions.ST_HillShade(*args, **kwargs)`

Returns the hypothetical illumination of an elevation raster band using provided azimuth, altitude, brightness and scale inputs.

see [http://postgis.net/docs/RT\\_ST\\_HillShade.html](http://postgis.net/docs/RT_ST_HillShade.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_Histogram(*args, **kwargs)`

Returns a set of record summarizing a raster or raster coverage data distribution separate bin ranges. Number of bins are autocomputed if not specified.

see [http://postgis.net/docs/RT\\_ST\\_Histogram.html](http://postgis.net/docs/RT_ST_Histogram.html)

**class** `geoalchemy2.functions.ST_InteriorRingN(*args, **kwargs)`

Returns the Nth interior ring (hole) of a Polygon.

see [http://postgis.net/docs/ST\\_InteriorRingN.html](http://postgis.net/docs/ST_InteriorRingN.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_InterpolatePoint(*args, **kwargs)`

Return the value of the measure dimension of a geometry at the point closed to the provided point.

see [http://postgis.net/docs/ST\\_InterpolatePoint.html](http://postgis.net/docs/ST_InterpolatePoint.html)

**class** `geoalchemy2.functions.ST_Intersection(*args, **kwargs)`

[geometry] (T) Returns a geometry that represents the shared portion of geomA and geomB. OR [raster] Returns a raster or a set of geometry-pixelvalue pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.

see [http://postgis.net/docs/ST\\_Intersection.html](http://postgis.net/docs/ST_Intersection.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_Intersects(*args, **kwargs)`

[geometry] Returns TRUE if the Geometries/Geography “spatially intersect in 2D” - (share any portion of space) and FALSE if they don’t (they are Disjoint). For geography tolerance is 0.00001 meters (so any points that close are considered to intersect) OR [raster] Return true if raster rastA spatially intersects raster rastB.

see [http://postgis.net/docs/ST\\_Intersects.html](http://postgis.net/docs/ST_Intersects.html)

**class** `geoalchemy2.functions.ST_InvDistWeight4ma(*args, **kwargs)`

Raster processing function that interpolates a pixel’s value from the pixel’s neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_InvDistWeight4ma.html](http://postgis.net/docs/RT_ST_InvDistWeight4ma.html)

**class** geoalchemy2.functions.**ST\_IsClosed**(\*args, \*\*kwargs)

Tests if a LineStrings's start and end points are coincident. For a PolyhedralSurface tests if it is closed (volumetric).

see [http://postgis.net/docs/ST\\_IsClosed.html](http://postgis.net/docs/ST_IsClosed.html)

**class** geoalchemy2.functions.**ST\_IsCollection**(\*args, \*\*kwargs)

Tests if a geometry is a geometry collection type.

see [http://postgis.net/docs/ST\\_IsCollection.html](http://postgis.net/docs/ST_IsCollection.html)

**class** geoalchemy2.functions.**ST\_IsEmpty**(\*args, \*\*kwargs)

[geometry] Tests if a geometry is empty. OR [raster] Returns true if the raster is empty (width = 0 and height = 0). Otherwise, returns false.

see [http://postgis.net/docs/ST\\_IsEmpty.html](http://postgis.net/docs/ST_IsEmpty.html)

**class** geoalchemy2.functions.**ST\_IsPlanar**(\*args, \*\*kwargs)

Check if a surface is or not planar

see [http://postgis.net/docs/ST\\_IsPlanar.html](http://postgis.net/docs/ST_IsPlanar.html)

**class** geoalchemy2.functions.**ST\_IsPolygonCCW**(\*args, \*\*kwargs)

Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

see [http://postgis.net/docs/ST\\_IsPolygonCCW.html](http://postgis.net/docs/ST_IsPolygonCCW.html)

**class** geoalchemy2.functions.**ST\_IsPolygonCW**(\*args, \*\*kwargs)

Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

see [http://postgis.net/docs/ST\\_IsPolygonCW.html](http://postgis.net/docs/ST_IsPolygonCW.html)

**class** geoalchemy2.functions.**ST\_IsRing**(\*args, \*\*kwargs)

Tests if a LineString is closed and simple.

see [http://postgis.net/docs/ST\\_IsRing.html](http://postgis.net/docs/ST_IsRing.html)

**class** geoalchemy2.functions.**ST\_IsSimple**(\*args, \*\*kwargs)

Tests if a geometry has no points of self-intersection or self-tangency.

see [http://postgis.net/docs/ST\\_IsSimple.html](http://postgis.net/docs/ST_IsSimple.html)

**class** geoalchemy2.functions.**ST\_IsSolid**(\*args, \*\*kwargs)

Test if the geometry is a solid. No validity check is performed.

see [http://postgis.net/docs/ST\\_IsSolid.html](http://postgis.net/docs/ST_IsSolid.html)

**class** geoalchemy2.functions.**ST\_IsValid**(\*args, \*\*kwargs)

Tests if a geometry is well-formed in 2D.

see [http://postgis.net/docs/ST\\_IsValid.html](http://postgis.net/docs/ST_IsValid.html)

**class** geoalchemy2.functions.**ST\_IsValidDetail**(\*args, \*\*kwargs)

Returns a valid\_detail row stating if a geometry is valid, and if not a reason why and a location.

see [http://postgis.net/docs/ST\\_IsValidDetail.html](http://postgis.net/docs/ST_IsValidDetail.html)

**class** geoalchemy2.functions.**ST\_IsValidReason**(\*args, \*\*kwargs)

Returns text stating if a geometry is valid, or a reason for invalidity.

see [http://postgis.net/docs/ST\\_IsValidReason.html](http://postgis.net/docs/ST_IsValidReason.html)

**class** `geoalchemy2.functions.ST_IsValidTrajectory(*args, **kwargs)`

Returns true if the geometry is a valid trajectory.

see [http://postgis.net/docs/ST\\_IsValidTrajectory.html](http://postgis.net/docs/ST_IsValidTrajectory.html)

**class** `geoalchemy2.functions.ST_Length(*args, **kwargs)`

Returns the 2D length of a linear geometry.

see [http://postgis.net/docs/ST\\_Length.html](http://postgis.net/docs/ST_Length.html)

**class** `geoalchemy2.functions.ST_Length2D(*args, **kwargs)`

Returns the 2D length of a linear geometry. Alias for `ST_Length`

see [http://postgis.net/docs/ST\\_Length2D.html](http://postgis.net/docs/ST_Length2D.html)

**class** `geoalchemy2.functions.ST_LengthSpheroid(*args, **kwargs)`

Returns the 2D or 3D length/perimeter of a lon/lat geometry on a spheroid.

see [http://postgis.net/docs/ST\\_LengthSpheroid.html](http://postgis.net/docs/ST_LengthSpheroid.html)

**class** `geoalchemy2.functions.ST_LineCrossingDirection(*args, **kwargs)`

Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.

see [http://postgis.net/docs/ST\\_LineCrossingDirection.html](http://postgis.net/docs/ST_LineCrossingDirection.html)

**class** `geoalchemy2.functions.ST_LineFromEncodedPolyline(*args, **kwargs)`

Creates a LineString from an Encoded Polyline.

see [http://postgis.net/docs/ST\\_LineFromEncodedPolyline.html](http://postgis.net/docs/ST_LineFromEncodedPolyline.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineFromMultiPoint(*args, **kwargs)`

Creates a LineString from a MultiPoint geometry.

see [http://postgis.net/docs/ST\\_LineFromMultiPoint.html](http://postgis.net/docs/ST_LineFromMultiPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineFromText(*args, **kwargs)`

Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to 0.

see [http://postgis.net/docs/ST\\_LineFromText.html](http://postgis.net/docs/ST_LineFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineFromWKB(*args, **kwargs)`

Makes a LINESTRING from WKB with the given SRID

see [http://postgis.net/docs/ST\\_LineFromWKB.html](http://postgis.net/docs/ST_LineFromWKB.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineInterpolatePoint(*args, **kwargs)`

Returns a point interpolated along a line. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.

see [http://postgis.net/docs/ST\\_LineInterpolatePoint.html](http://postgis.net/docs/ST_LineInterpolatePoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineInterpolatePoints(*args, **kwargs)`

Returns one or more points interpolated along a line.

see [http://postgis.net/docs/ST\\_LineInterpolatePoints.html](http://postgis.net/docs/ST_LineInterpolatePoints.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineLocatePoint(*args, **kwargs)`

Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total 2d line length.

see [http://postgis.net/docs/ST\\_LineLocatePoint.html](http://postgis.net/docs/ST_LineLocatePoint.html)

**class** `geoalchemy2.functions.ST_LineMerge(*args, **kwargs)`

Return a (set of) LineString(s) formed by sewing together a MULTILINESTRING.

see [http://postgis.net/docs/ST\\_LineMerge.html](http://postgis.net/docs/ST_LineMerge.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineSubstring(*args, **kwargs)`

Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.

see [http://postgis.net/docs/ST\\_LineSubstring.html](http://postgis.net/docs/ST_LineSubstring.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_LineToCurve(*args, **kwargs)`

Converts a LINESTRING/POLYGON to a CIRCULARSTRING, CURVEPOLYGON

see [http://postgis.net/docs/ST\\_LineToCurve.html](http://postgis.net/docs/ST_LineToCurve.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*



**class** `geoalchemy2.functions.ST_LinestringFromWKB(*args, **kwargs)`

Makes a geometry from WKB with the given SRID.

see [http://postgis.net/docs/ST\\_LinestringFromWKB.html](http://postgis.net/docs/ST_LinestringFromWKB.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_LocateAlong(*args, **kwargs)`

Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

see [http://postgis.net/docs/ST\\_LocateAlong.html](http://postgis.net/docs/ST_LocateAlong.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_LocateBetween(*args, **kwargs)`

Return a derived geometry collection value with elements that match the specified range of measures inclusively.

see [http://postgis.net/docs/ST\\_LocateBetween.html](http://postgis.net/docs/ST_LocateBetween.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_LocateBetweenElevations(*args, **kwargs)`

Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively.

see [http://postgis.net/docs/ST\\_LocateBetweenElevations.html](http://postgis.net/docs/ST_LocateBetweenElevations.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_LongestLine(*args, **kwargs)`

Returns the 2D longest line between two geometries.

see [http://postgis.net/docs/ST\\_LongestLine.html](http://postgis.net/docs/ST_LongestLine.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_M(*args, **kwargs)`

Returns the M coordinate of a Point.

see [http://postgis.net/docs/ST\\_M.html](http://postgis.net/docs/ST_M.html)

**class** `geoalchemy2.functions.ST_MLineFromText(*args, **kwargs)`

Return a specified ST\_MultiLineString value from WKT representation.

see [http://postgis.net/docs/ST\\_MLineFromText.html](http://postgis.net/docs/ST_MLineFromText.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MPointFromText**(\*args, \*\*kwargs)

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see [http://postgis.net/docs/ST\\_MPointFromText.html](http://postgis.net/docs/ST_MPointFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MPolyFromText**(\*args, \*\*kwargs)

Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see [http://postgis.net/docs/ST\\_MPolyFromText.html](http://postgis.net/docs/ST_MPolyFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakeBox2D**(\*args, \*\*kwargs)

Creates a BOX2D defined by two 2D point geometries.

see [http://postgis.net/docs/ST\\_MakeBox2D.html](http://postgis.net/docs/ST_MakeBox2D.html)

**class** geoalchemy2.functions.**ST\_MakeEmptyCoverage**(\*args, \*\*kwargs)

Cover georeferenced area with a grid of empty raster tiles.

see [http://postgis.net/docs/RT\\_ST\\_MakeEmptyCoverage.html](http://postgis.net/docs/RT_ST_MakeEmptyCoverage.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** geoalchemy2.functions.**ST\_MakeEmptyRaster**(\*args, \*\*kwargs)

Returns an empty raster (having no bands) of given dimensions (width & height), upperleft X and Y, pixel size and rotation (scalex, scaley, skewx & skewy) and reference system (srid). If a raster is passed in, returns a new raster with the same size, alignment and SRID. If srid is left out, the spatial ref is set to unknown (0).

see [http://postgis.net/docs/RT\\_ST\\_MakeEmptyRaster.html](http://postgis.net/docs/RT_ST_MakeEmptyRaster.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** geoalchemy2.functions.**ST\_MakeEnvelope**(\*args, \*\*kwargs)

Creates a rectangular Polygon from minimum and maximum coordinates.

see [http://postgis.net/docs/ST\\_MakeEnvelope.html](http://postgis.net/docs/ST_MakeEnvelope.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakeLine**(\*args, \*\*kwargs)

Creates a LineString from Point, MultiPoint, or LineString geometries.

see [http://postgis.net/docs/ST\\_MakeLine.html](http://postgis.net/docs/ST_MakeLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakePoint**(\*args, \*\*kwargs)

Creates a 2D, 3DZ or 4D Point.

see [http://postgis.net/docs/ST\\_MakePoint.html](http://postgis.net/docs/ST_MakePoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakePointM**(\*args, \*\*kwargs)

Creates a Point from X, Y and M values.

see [http://postgis.net/docs/ST\\_MakePointM.html](http://postgis.net/docs/ST_MakePointM.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakePolygon**(\*args, \*\*kwargs)

Creates a Polygon from a shell and optional list of holes.

see [http://postgis.net/docs/ST\\_MakePolygon.html](http://postgis.net/docs/ST_MakePolygon.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakeSolid**(\*args, \*\*kwargs)

Cast the geometry into a solid. No check is performed. To obtain a valid solid, the input geometry must be a closed Polyhedral Surface or a closed TIN.

see [http://postgis.net/docs/ST\\_MakeSolid.html](http://postgis.net/docs/ST_MakeSolid.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_MakeValid**(\*args, \*\*kwargs)

Attempts to make an invalid geometry valid without losing vertices.

see [http://postgis.net/docs/ST\\_MakeValid.html](http://postgis.net/docs/ST_MakeValid.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_MapAlgebra(*args, **kwargs)`

[raster] Callback function version - Returns a one-band raster given one or more input rasters, band indexes and one user-specified callback function. OR [raster] Expression version - Returns a one-band raster given one or two input rasters, band indexes and one or more user-specified SQL expressions.

see [http://postgis.net/docs/RT\\_ST\\_MapAlgebra.html](http://postgis.net/docs/RT_ST_MapAlgebra.html)

**class** `geoalchemy2.functions.ST_MapAlgebraExpr(*args, **kwargs)`

[raster] 1 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified. OR [raster] 2 raster band version: Creates a new one band raster formed by applying a valid PostgreSQL algebraic operation on the two input raster bands and of pixeltype provided. band 1 of each raster is assumed if no band numbers are specified. The resulting raster will be aligned (scale, skew and pixel corners) on the grid defined by the first raster and have its extent defined by the “extenttype” parameter. Values for “extenttype” can be: INTERSECTION, UNION, FIRST, SECOND.

see [http://postgis.net/docs/RT\\_ST\\_MapAlgebraExpr.html](http://postgis.net/docs/RT_ST_MapAlgebraExpr.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_MapAlgebraFct(*args, **kwargs)`

[raster] 1 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the input raster band and of pixeltype provided. Band 1 is assumed if no band is specified. OR [raster] 2 band version - Creates a new one band raster formed by applying a valid PostgreSQL function on the 2 input raster bands and of pixeltype provided. Band 1 is assumed if no band is specified. Extent type defaults to INTERSECTION if not specified.

see [http://postgis.net/docs/RT\\_ST\\_MapAlgebraFct.html](http://postgis.net/docs/RT_ST_MapAlgebraFct.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_MapAlgebraFctNgb(*args, **kwargs)`

1-band version: Map Algebra Nearest Neighbor using user-defined PostgreSQL function. Return a raster which values are the result of a PLPGSQL user function involving a neighborhood of values from the input raster band.

see [http://postgis.net/docs/RT\\_ST\\_MapAlgebraFctNgb.html](http://postgis.net/docs/RT_ST_MapAlgebraFctNgb.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_Max4ma(*args, **kwargs)`

Raster processing function that calculates the maximum pixel value in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Max4ma.html](http://postgis.net/docs/RT_ST_Max4ma.html)

**class** `geoalchemy2.functions.ST_MaxDistance(*args, **kwargs)`

Returns the 2D largest distance between two geometries in projected units.

see [http://postgis.net/docs/ST\\_MaxDistance.html](http://postgis.net/docs/ST_MaxDistance.html)

**class** `geoalchemy2.functions.ST_Mean4ma(*args, **kwargs)`

Raster processing function that calculates the mean pixel value in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Mean4ma.html](http://postgis.net/docs/RT_ST_Mean4ma.html)

**class** `geoalchemy2.functions.ST_MemSize(*args, **kwargs)`

[geometry] Returns the amount of memory space a geometry takes. OR [raster] Returns the amount of space (in bytes) the raster takes.

see [http://postgis.net/docs/ST\\_MemSize.html](http://postgis.net/docs/ST_MemSize.html)

**class** `geoalchemy2.functions.ST_MemUnion(*args, **kwargs)`

Same as ST\_Union, only memory-friendly (uses less memory and more processor time).

see [http://postgis.net/docs/ST\\_MemUnion.html](http://postgis.net/docs/ST_MemUnion.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_MetaData(*args, **kwargs)`

Returns basic meta data about a raster object such as pixel size, rotation (skew), upper, lower left, etc.

see [http://postgis.net/docs/RT\\_ST\\_MetaData.html](http://postgis.net/docs/RT_ST_MetaData.html)

**class** `geoalchemy2.functions.ST_Min4ma(*args, **kwargs)`

Raster processing function that calculates the minimum pixel value in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Min4ma.html](http://postgis.net/docs/RT_ST_Min4ma.html)

**class** `geoalchemy2.functions.ST_MinConvexHull(*args, **kwargs)`

Return the convex hull geometry of the raster excluding NODATA pixels.

see [http://postgis.net/docs/RT\\_ST\\_MinConvexHull.html](http://postgis.net/docs/RT_ST_MinConvexHull.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_MinDist4ma(*args, **kwargs)`

Raster processing function that returns the minimum distance (in number of pixels) between the pixel of interest and a neighboring pixel with value.

see [http://postgis.net/docs/RT\\_ST\\_MinDist4ma.html](http://postgis.net/docs/RT_ST_MinDist4ma.html)

**class** `geoalchemy2.functions.ST_MinPossibleValue(*args, **kwargs)`

Returns the minimum value this pixeltype can store.

see [http://postgis.net/docs/ST\\_MinPossibleValue.html](http://postgis.net/docs/ST_MinPossibleValue.html)

**class** `geoalchemy2.functions.ST_MinimumBoundingCircle(*args, **kwargs)`

Returns the smallest circle polygon that can fully contain a geometry. Default uses 48 segments per quarter circle.

see [http://postgis.net/docs/ST\\_MinimumBoundingCircle.html](http://postgis.net/docs/ST_MinimumBoundingCircle.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_MinimumBoundingRadius(*args, **kwargs)`

Returns the center point and radius of the smallest circle that can fully contain a geometry.

see [http://postgis.net/docs/ST\\_MinimumBoundingRadius.html](http://postgis.net/docs/ST_MinimumBoundingRadius.html)

**class** `geoalchemy2.functions.ST_MinimumClearance(*args, **kwargs)`

Returns the minimum clearance of a geometry, a measure of a geometry's robustness.

see [http://postgis.net/docs/ST\\_MinimumClearance.html](http://postgis.net/docs/ST_MinimumClearance.html)

**class** `geoalchemy2.functions.ST_MinimumClearanceLine(*args, **kwargs)`

Returns the two-point LineString spanning a geometry's minimum clearance.

see [http://postgis.net/docs/ST\\_MinimumClearanceLine.html](http://postgis.net/docs/ST_MinimumClearanceLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_MinkowskiSum(*args, **kwargs)`

Performs Minkowski sum

see [http://postgis.net/docs/ST\\_MinkowskiSum.html](http://postgis.net/docs/ST_MinkowskiSum.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Multi(*args, **kwargs)`

Return the geometry as a MULTI\* geometry.

see [http://postgis.net/docs/ST\\_Multi.html](http://postgis.net/docs/ST_Multi.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_NDims(*args, **kwargs)`

Returns the coordinate dimension of a geometry.

see [http://postgis.net/docs/ST\\_NDims.html](http://postgis.net/docs/ST_NDims.html)

**class** `geoalchemy2.functions.ST_NPoints(*args, **kwargs)`

Returns the number of points (vertices) in a geometry.

see [http://postgis.net/docs/ST\\_NPoints.html](http://postgis.net/docs/ST_NPoints.html)

**class** `geoalchemy2.functions.ST_NRings(*args, **kwargs)`

Returns the number of rings in a polygonal geometry.

see [http://postgis.net/docs/ST\\_NRings.html](http://postgis.net/docs/ST_NRings.html)

**class** `geoalchemy2.functions.ST_NearestValue(*args, **kwargs)`

Returns the nearest non-NODATA value of a given band's pixel specified by a columnx and rowy or a geometric point expressed in the same spatial reference coordinate system as the raster.

see [http://postgis.net/docs/RT\\_ST\\_NearestValue.html](http://postgis.net/docs/RT_ST_NearestValue.html)

**class** `geoalchemy2.functions.ST_Neighborhood(*args, **kwargs)`

Returns a 2-D double precision array of the non-NODATA values around a given band's pixel specified by either a columnX and rowY or a geometric point expressed in the same spatial reference coordinate system as the raster.

see [http://postgis.net/docs/RT\\_ST\\_Neighborhood.html](http://postgis.net/docs/RT_ST_Neighborhood.html)

**class** `geoalchemy2.functions.ST_Node(*args, **kwargs)`

Node a set of linestrings.

see [http://postgis.net/docs/ST\\_Node.html](http://postgis.net/docs/ST_Node.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Normalize(*args, **kwargs)`

Return the geometry in its canonical form.

see [http://postgis.net/docs/ST\\_Normalize.html](http://postgis.net/docs/ST_Normalize.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_NotSameAlignmentReason(*args, **kwargs)`

Returns text stating if rasters are aligned and if not aligned, a reason why.

see [http://postgis.net/docs/RT\\_ST\\_NotSameAlignmentReason.html](http://postgis.net/docs/RT_ST_NotSameAlignmentReason.html)

**class** `geoalchemy2.functions.ST_NumBands(*args, **kwargs)`

Returns the number of bands in the raster object.

see [http://postgis.net/docs/RT\\_ST\\_NumBands.html](http://postgis.net/docs/RT_ST_NumBands.html)

**class** `geoalchemy2.functions.ST_NumGeometries(*args, **kwargs)`

Returns the number of elements in a geometry collection.

see [http://postgis.net/docs/ST\\_NumGeometries.html](http://postgis.net/docs/ST_NumGeometries.html)

**class** `geoalchemy2.functions.ST_NumInteriorRing(*args, **kwargs)`

Returns the number of interior rings (holes) of a Polygon. Alias for ST\_NumInteriorRings

see [http://postgis.net/docs/ST\\_NumInteriorRing.html](http://postgis.net/docs/ST_NumInteriorRing.html)

**class** `geoalchemy2.functions.ST_NumInteriorRings(*args, **kwargs)`

Returns the number of interior rings (holes) of a Polygon.

see [http://postgis.net/docs/ST\\_NumInteriorRings.html](http://postgis.net/docs/ST_NumInteriorRings.html)

**class** `geoalchemy2.functions.ST_NumPatches(*args, **kwargs)`

Return the number of faces on a Polyhedral Surface. Will return null for non-polyhedral geometries.

see [http://postgis.net/docs/ST\\_NumPatches.html](http://postgis.net/docs/ST_NumPatches.html)

**class** `geoalchemy2.functions.ST_NumPoints(*args, **kwargs)`

Returns the number of points in a LineString or CircularString.

see [http://postgis.net/docs/ST\\_NumPoints.html](http://postgis.net/docs/ST_NumPoints.html)

**class** `geoalchemy2.functions.ST_OffsetCurve(*args, **kwargs)`

Return an offset line at a given distance and side from an input line. Useful for computing parallel lines about a center line

see [http://postgis.net/docs/ST\\_OffsetCurve.html](http://postgis.net/docs/ST_OffsetCurve.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_OrderingEquals(*args, **kwargs)`

Returns true if the given geometries represent the same geometry and points are in the same directional order.

see [http://postgis.net/docs/ST\\_OrderingEquals.html](http://postgis.net/docs/ST_OrderingEquals.html)

**class** `geoalchemy2.functions.ST_Orientation(*args, **kwargs)`

Determine surface orientation

see [http://postgis.net/docs/ST\\_Orientation.html](http://postgis.net/docs/ST_Orientation.html)

**class** `geoalchemy2.functions.ST_OrientedEnvelope(*args, **kwargs)`

Returns a minimum rotated rectangle enclosing a geometry.

see [http://postgis.net/docs/ST\\_OrientedEnvelope.html](http://postgis.net/docs/ST_OrientedEnvelope.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Overlaps(*args, **kwargs)`

[geometry] Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other. OR [raster] Return true if raster rastA and rastB intersect but one does not completely contain the other.

see [http://postgis.net/docs/ST\\_Overlaps.html](http://postgis.net/docs/ST_Overlaps.html)

**class** `geoalchemy2.functions.ST_PatchN(*args, **kwargs)`

Returns the Nth geometry (face) of a PolyhedralSurface.

see [http://postgis.net/docs/ST\\_PatchN.html](http://postgis.net/docs/ST_PatchN.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Perimeter(*args, **kwargs)`

Returns the length of the boundary of a polygonal geometry or geography.

see [http://postgis.net/docs/ST\\_Perimeter.html](http://postgis.net/docs/ST_Perimeter.html)

**class** `geoalchemy2.functions.ST_Perimeter2D(*args, **kwargs)`

Returns the 2D perimeter of a polygonal geometry. Alias for ST\_Perimeter.

see [http://postgis.net/docs/ST\\_Perimeter2D.html](http://postgis.net/docs/ST_Perimeter2D.html)

**class** `geoalchemy2.functions.ST_PixelAsCentroid(*args, **kwargs)`

Returns the centroid (point geometry) of the area represented by a pixel.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsCentroid.html](http://postgis.net/docs/RT_ST_PixelAsCentroid.html)

Return type: *geoalchemy2.types.Geometry*.



**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PixelAsCentroids(*args, **kwargs)`

Returns the centroid (point geometry) for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The point geometry is the centroid of the area represented by a pixel.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsCentroids.html](http://postgis.net/docs/RT_ST_PixelAsCentroids.html)

**class** `geoalchemy2.functions.ST_PixelAsPoint(*args, **kwargs)`

Returns a point geometry of the pixel's upper-left corner.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsPoint.html](http://postgis.net/docs/RT_ST_PixelAsPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PixelAsPoints(*args, **kwargs)`

Returns a point geometry for each pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel. The coordinates of the point geometry are of the pixel's upper-left corner.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsPoints.html](http://postgis.net/docs/RT_ST_PixelAsPoints.html)

**class** `geoalchemy2.functions.ST_PixelAsPolygon(*args, **kwargs)`

Returns the polygon geometry that bounds the pixel for a particular row and column.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsPolygon.html](http://postgis.net/docs/RT_ST_PixelAsPolygon.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PixelAsPolygons(*args, **kwargs)`

Returns the polygon geometry that bounds every pixel of a raster band along with the value, the X and the Y raster coordinates of each pixel.

see [http://postgis.net/docs/RT\\_ST\\_PixelAsPolygons.html](http://postgis.net/docs/RT_ST_PixelAsPolygons.html)

**class** `geoalchemy2.functions.ST_PixelHeight(*args, **kwargs)`

Returns the pixel height in geometric units of the spatial reference system.

see [http://postgis.net/docs/RT\\_ST\\_PixelHeight.html](http://postgis.net/docs/RT_ST_PixelHeight.html)

**class** `geoalchemy2.functions.ST_PixelOfValue(*args, **kwargs)`

Get the columnx, rowy coordinates of the pixel whose value equals the search value.

see [http://postgis.net/docs/RT\\_ST\\_PixelOfValue.html](http://postgis.net/docs/RT_ST_PixelOfValue.html)

**class** `geoalchemy2.functions.ST_PixelWidth(*args, **kwargs)`

Returns the pixel width in geometric units of the spatial reference system.

see [http://postgis.net/docs/RT\\_ST\\_PixelWidth.html](http://postgis.net/docs/RT_ST_PixelWidth.html)

**class** `geoalchemy2.functions.ST_Point(*args, **kwargs)`

Creates a Point with the given coordinate values. Alias for `ST_MakePoint`.

see [http://postgis.net/docs/ST\\_Point.html](http://postgis.net/docs/ST_Point.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PointFromGeoHash(*args, **kwargs)`

Return a point from a GeoHash string.

see [http://postgis.net/docs/ST\\_PointFromGeoHash.html](http://postgis.net/docs/ST_PointFromGeoHash.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PointFromText(*args, **kwargs)`

Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.

see [http://postgis.net/docs/ST\\_PointFromText.html](http://postgis.net/docs/ST_PointFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PointFromWKB(*args, **kwargs)`

Makes a geometry from WKB with the given SRID

see [http://postgis.net/docs/ST\\_PointFromWKB.html](http://postgis.net/docs/ST_PointFromWKB.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PointInsideCircle(*args, **kwargs)`

Is the point geometry inside the circle defined by center\_x, center\_y, radius

see [http://postgis.net/docs/ST\\_PointInsideCircle.html](http://postgis.net/docs/ST_PointInsideCircle.html)

**class** `geoalchemy2.functions.ST_PointN(*args, **kwargs)`

Returns the Nth point in the first LineString or circular LineString in a geometry.

see [http://postgis.net/docs/ST\\_PointN.html](http://postgis.net/docs/ST_PointN.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PointOnSurface(*args, **kwargs)`

Returns a POINT guaranteed to lie on the surface.

see [http://postgis.net/docs/ST\\_PointOnSurface.html](http://postgis.net/docs/ST_PointOnSurface.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Points(*args, **kwargs)`

Returns a MultiPoint containing all the coordinates of a geometry.

see [http://postgis.net/docs/ST\\_Points.html](http://postgis.net/docs/ST_Points.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Polygon(*args, **kwargs)`

[geometry] Creates a Polygon from a LineString with a specified SRID. OR [raster] Returns a multipolygon geometry formed by the union of pixels that have a pixel value that is not no data value. If no band number is specified, band num defaults to 1.

see [http://postgis.net/docs/ST\\_Polygon.html](http://postgis.net/docs/ST_Polygon.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_PolygonFromText(*args, **kwargs)`

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

see [http://postgis.net/docs/ST\\_PolygonFromText.html](http://postgis.net/docs/ST_PolygonFromText.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Polygonize(*args, **kwargs)`

Aggregate. Creates a GeometryCollection containing possible polygons formed from the constituent linework of a set of geometries.

see [http://postgis.net/docs/ST\\_Polygonize.html](http://postgis.net/docs/ST_Polygonize.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Project(*args, **kwargs)`

Returns a point projected from a start point by a distance and bearing (azimuth).

see [http://postgis.net/docs/ST\\_Project.html](http://postgis.net/docs/ST_Project.html)

Return type: *geoalchemy2.types.Geography*.

**type**

alias of *Geography*

**class** `geoalchemy2.functions.ST_Quantile(*args, **kwargs)`

Compute quantiles for a raster or raster table coverage in the context of the sample or population. Thus, a value could be examined to be at the raster's 25%, 50%, 75% percentile.

see [http://postgis.net/docs/RT\\_ST\\_Quantile.html](http://postgis.net/docs/RT_ST_Quantile.html)

**class** `geoalchemy2.functions.ST_QuantizeCoordinates(*args, **kwargs)`

Sets least significant bits of coordinates to zero

see [http://postgis.net/docs/ST\\_QuantizeCoordinates.html](http://postgis.net/docs/ST_QuantizeCoordinates.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Range4ma(*args, **kwargs)`

Raster processing function that calculates the range of pixel values in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Range4ma.html](http://postgis.net/docs/RT_ST_Range4ma.html)

**class** `geoalchemy2.functions.ST_RastFromHexWKB(*args, **kwargs)`

Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

see [http://postgis.net/docs/RT\\_ST\\_RastFromHexWKB.html](http://postgis.net/docs/RT_ST_RastFromHexWKB.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_RastFromWKB(*args, **kwargs)`

Return a raster value from a Well-Known Binary (WKB) raster.

see [http://postgis.net/docs/RT\\_ST\\_RastFromWKB.html](http://postgis.net/docs/RT_ST_RastFromWKB.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_RasterToWorldCoord(*args, **kwargs)`

Returns the raster's upper left corner as geometric X and Y (longitude and latitude) given a column and row. Column and row starts at 1.

see [http://postgis.net/docs/RT\\_ST\\_RasterToWorldCoord.html](http://postgis.net/docs/RT_ST_RasterToWorldCoord.html)

**class** `geoalchemy2.functions.ST_RasterToWorldCoordX(*args, **kwargs)`

Returns the geometric X coordinate upper left of a raster, column and row. Numbering of columns and rows starts at 1.

see [http://postgis.net/docs/RT\\_ST\\_RasterToWorldCoordX.html](http://postgis.net/docs/RT_ST_RasterToWorldCoordX.html)

**class** `geoalchemy2.functions.ST_RasterToWorldCoordY(*args, **kwargs)`

Returns the geometric Y coordinate upper left corner of a raster, column and row. Numbering of columns and rows starts at 1.

see [http://postgis.net/docs/RT\\_ST\\_RasterToWorldCoordY.html](http://postgis.net/docs/RT_ST_RasterToWorldCoordY.html)

**class** `geoalchemy2.functions.ST_Reclass(*args, **kwargs)`

Creates a new raster composed of band types reclassified from original. The nband is the band to be changed. If nband is not specified assumed to be 1. All other bands are returned unchanged. Use case: convert a 16BUI band to a 8BUI and so forth for simpler rendering as viewable formats.

see [http://postgis.net/docs/RT\\_ST\\_Reclass.html](http://postgis.net/docs/RT_ST_Reclass.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_Relate(*args, **kwargs)`

Returns true if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries.

see [http://postgis.net/docs/ST\\_Relate.html](http://postgis.net/docs/ST_Relate.html)

**class** `geoalchemy2.functions.ST_RelateMatch(*args, **kwargs)`

Returns true if `intersectionMatrixPattern1` implies `intersectionMatrixPattern2`

see [http://postgis.net/docs/ST\\_RelateMatch.html](http://postgis.net/docs/ST_RelateMatch.html)

**class** `geoalchemy2.functions.ST_RemovePoint(*args, **kwargs)`

Remove point from a linestring.

see [http://postgis.net/docs/ST\\_RemovePoint.html](http://postgis.net/docs/ST_RemovePoint.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_RemoveRepeatedPoints(*args, **kwargs)`

Returns a version of the given geometry with duplicated points removed.

see [http://postgis.net/docs/ST\\_RemoveRepeatedPoints.html](http://postgis.net/docs/ST_RemoveRepeatedPoints.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_Resample(*args, **kwargs)`

Resample a raster using a specified resampling algorithm, new dimensions, an arbitrary grid corner and a set of raster georeferencing attributes defined or borrowed from another raster.

see [http://postgis.net/docs/RT\\_ST\\_Resample.html](http://postgis.net/docs/RT_ST_Resample.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_Rescale(*args, **kwargs)`

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see [http://postgis.net/docs/RT\\_ST\\_Rescale.html](http://postgis.net/docs/RT_ST_Rescale.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_Resize(*args, **kwargs)`

Resize a raster to a new width/height

see [http://postgis.net/docs/RT\\_ST\\_Resize.html](http://postgis.net/docs/RT_ST_Resize.html)

Return type: `geoalchemy2.types.Raster`.

**type**

alias of `Raster`

**class** `geoalchemy2.functions.ST_Reskew(*args, **kwargs)`

Resample a raster by adjusting only its skew (or rotation parameters). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see [http://postgis.net/docs/RT\\_ST\\_Reskew.html](http://postgis.net/docs/RT_ST_Reskew.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** *geoalchemy2.functions.ST\_Retile(\*args, \*\*kwargs)*

Return a set of configured tiles from an arbitrarily tiled raster coverage.

see [http://postgis.net/docs/RT\\_ST\\_Retile.html](http://postgis.net/docs/RT_ST_Retile.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** *geoalchemy2.functions.ST\_Reverse(\*args, \*\*kwargs)*

Return the geometry with vertex order reversed.

see [http://postgis.net/docs/ST\\_Reverse.html](http://postgis.net/docs/ST_Reverse.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** *geoalchemy2.functions.ST\_Rotate(\*args, \*\*kwargs)*

Rotates a geometry about an origin point.

see [http://postgis.net/docs/ST\\_Rotate.html](http://postgis.net/docs/ST_Rotate.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** *geoalchemy2.functions.ST\_RotateX(\*args, \*\*kwargs)*

Rotates a geometry about the X axis.

see [http://postgis.net/docs/ST\\_RotateX.html](http://postgis.net/docs/ST_RotateX.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** *geoalchemy2.functions.ST\_RotateY(\*args, \*\*kwargs)*

Rotates a geometry about the Y axis.

see [http://postgis.net/docs/ST\\_RotateY.html](http://postgis.net/docs/ST_RotateY.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** *geoalchemy2.functions.ST\_RotateZ(\*args, \*\*kwargs)*

Rotates a geometry about the Z axis.

see [http://postgis.net/docs/ST\\_RotateZ.html](http://postgis.net/docs/ST_RotateZ.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Rotation(*args, **kwargs)`

Returns the rotation of the raster in radian.

see [http://postgis.net/docs/RT\\_ST\\_Rotation.html](http://postgis.net/docs/RT_ST_Rotation.html)

**class** `geoalchemy2.functions.ST_Roughness(*args, **kwargs)`

Returns a raster with the calculated “roughness” of a DEM.

see [http://postgis.net/docs/RT\\_ST\\_Roughness.html](http://postgis.net/docs/RT_ST_Roughness.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SRID(*args, **kwargs)`

[geometry] Returns the spatial reference identifier for the ST\_Geometry as defined in spatial\_ref\_sys table. OR  
[raster] Returns the spatial reference identifier of the raster as defined in spatial\_ref\_sys table.

see [http://postgis.net/docs/ST\\_SRID.html](http://postgis.net/docs/ST_SRID.html)

**class** `geoalchemy2.functions.ST_SameAlignment(*args, **kwargs)`

Returns true if rasters have same skew, scale, spatial ref, and offset (pixels can be put on same grid without cutting into pixels) and false if they don’t with notice detailing issue.

see [http://postgis.net/docs/RT\\_ST\\_SameAlignment.html](http://postgis.net/docs/RT_ST_SameAlignment.html)

**class** `geoalchemy2.functions.ST_Scale(*args, **kwargs)`

Scales a geometry by given factors.

see [http://postgis.net/docs/ST\\_Scale.html](http://postgis.net/docs/ST_Scale.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ScaleX(*args, **kwargs)`

Returns the X component of the pixel width in units of coordinate reference system.

see [http://postgis.net/docs/RT\\_ST\\_ScaleX.html](http://postgis.net/docs/RT_ST_ScaleX.html)

**class** `geoalchemy2.functions.ST_ScaleY(*args, **kwargs)`

Returns the Y component of the pixel height in units of coordinate reference system.

see [http://postgis.net/docs/RT\\_ST\\_ScaleY.html](http://postgis.net/docs/RT_ST_ScaleY.html)

**class** `geoalchemy2.functions.ST_Segmentize(*args, **kwargs)`

Return a modified geometry/geography having no segment longer than the given distance.

see [http://postgis.net/docs/ST\\_Segmentize.html](http://postgis.net/docs/ST_Segmentize.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SetBandIndex(*args, **kwargs)`

Update the external band number of an out-db band

see [http://postgis.net/docs/RT\\_ST\\_SetBandIndex.html](http://postgis.net/docs/RT_ST_SetBandIndex.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetBandIsNoData(*args, **kwargs)`

Sets the isnodata flag of the band to TRUE.

see [http://postgis.net/docs/RT\\_ST\\_SetBandIsNoData.html](http://postgis.net/docs/RT_ST_SetBandIsNoData.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetBandNoDataValue(*args, **kwargs)`

Sets the value for the given band that represents no data. Band 1 is assumed if no band is specified. To mark a band as having no nodata value, set the nodata value = NULL.

see [http://postgis.net/docs/RT\\_ST\\_SetBandNoDataValue.html](http://postgis.net/docs/RT_ST_SetBandNoDataValue.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetBandPath(*args, **kwargs)`

Update the external path and band number of an out-db band

see [http://postgis.net/docs/RT\\_ST\\_SetBandPath.html](http://postgis.net/docs/RT_ST_SetBandPath.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetEffectiveArea(*args, **kwargs)`

Sets the effective area for each vertex, storing the value in the M ordinate. A simplified geometry can then be generated by filtering on the M ordinate.

see [http://postgis.net/docs/ST\\_SetEffectiveArea.html](http://postgis.net/docs/ST_SetEffectiveArea.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SetGeoReference(*args, **kwargs)`

Set Georeference 6 georeference parameters in a single call. Numbers should be separated by white space. Accepts inputs in GDAL or ESRI format. Default is GDAL.

see [http://postgis.net/docs/RT\\_ST\\_SetGeoReference.html](http://postgis.net/docs/RT_ST_SetGeoReference.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*



**class** `geoalchemy2.functions.ST_SetPoint(*args, **kwargs)`

Replace point of a linestring with a given point.

see [http://postgis.net/docs/ST\\_SetPoint.html](http://postgis.net/docs/ST_SetPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SetRotation(*args, **kwargs)`

Set the rotation of the raster in radian.

see [http://postgis.net/docs/RT\\_ST\\_SetRotation.html](http://postgis.net/docs/RT_ST_SetRotation.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetSRID(*args, **kwargs)`

[geometry] Set the SRID on a geometry to a particular integer value. OR [raster] Sets the SRID of a raster to a particular integer srid defined in the spatial\_ref\_sys table.

see [http://postgis.net/docs/ST\\_SetSRID.html](http://postgis.net/docs/ST_SetSRID.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SetScale(*args, **kwargs)`

Sets the X and Y size of pixels in units of coordinate reference system. Number units/pixel width/height.

see [http://postgis.net/docs/RT\\_ST\\_SetScale.html](http://postgis.net/docs/RT_ST_SetScale.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetSkew(*args, **kwargs)`

Sets the georeference X and Y skew (or rotation parameter). If only one is passed in, sets X and Y to the same value.

see [http://postgis.net/docs/RT\\_ST\\_SetSkew.html](http://postgis.net/docs/RT_ST_SetSkew.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetUpperLeft(*args, **kwargs)`

Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

see [http://postgis.net/docs/RT\\_ST\\_SetUpperLeft.html](http://postgis.net/docs/RT_ST_SetUpperLeft.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetValue(*args, **kwargs)`

Returns modified raster resulting from setting the value of a given band in a given columnx, rowy pixel or the pixels that intersect a particular geometry. Band numbers start at 1 and assumed to be 1 if not specified.

see [http://postgis.net/docs/RT\\_ST\\_SetValue.html](http://postgis.net/docs/RT_ST_SetValue.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SetValues(*args, **kwargs)`

Returns modified raster resulting from setting the values of a given band.

see [http://postgis.net/docs/RT\\_ST\\_SetValues.html](http://postgis.net/docs/RT_ST_SetValues.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_SharedPaths(*args, **kwargs)`

Returns a collection containing paths shared by the two input linestrings/multilinestrings.

see [http://postgis.net/docs/ST\\_SharedPaths.html](http://postgis.net/docs/ST_SharedPaths.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ShiftLongitude(*args, **kwargs)`

Toggle geometry coordinates between -180..180 and 0..360 ranges.

see [http://postgis.net/docs/ST\\_Shift\\_Longitude.html](http://postgis.net/docs/ST_Shift_Longitude.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_ShortestLine(*args, **kwargs)`

Returns the 2D shortest line between two geometries

see [http://postgis.net/docs/ST\\_ShortestLine.html](http://postgis.net/docs/ST_ShortestLine.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Simplify(*args, **kwargs)`

Returns a “simplified” version of the given geometry using the Douglas-Peucker algorithm.

see [http://postgis.net/docs/ST\\_Simplify.html](http://postgis.net/docs/ST_Simplify.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SimplifyPreserveTopology(*args, **kwargs)`

Returns a “simplified” version of the given geometry using the Douglas-Peucker algorithm. Will avoid creating derived geometries (polygons in particular) that are invalid.

see [http://postgis.net/docs/ST\\_SimplifyPreserveTopology.html](http://postgis.net/docs/ST_SimplifyPreserveTopology.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SimplifyVW(*args, **kwargs)`

Returns a “simplified” version of the given geometry using the Visvalingam-Whyatt algorithm

see [http://postgis.net/docs/ST\\_SimplifyVW.html](http://postgis.net/docs/ST_SimplifyVW.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SkewX(*args, **kwargs)`

Returns the georeference X skew (or rotation parameter).

see [http://postgis.net/docs/RT\\_ST\\_SkewX.html](http://postgis.net/docs/RT_ST_SkewX.html)

**class** `geoalchemy2.functions.ST_SkewY(*args, **kwargs)`

Returns the georeference Y skew (or rotation parameter).

see [http://postgis.net/docs/RT\\_ST\\_SkewY.html](http://postgis.net/docs/RT_ST_SkewY.html)

**class** `geoalchemy2.functions.ST_Slope(*args, **kwargs)`

Returns the slope (in degrees by default) of an elevation raster band. Useful for analyzing terrain.

see [http://postgis.net/docs/RT\\_ST\\_Slope.html](http://postgis.net/docs/RT_ST_Slope.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_Snap(*args, **kwargs)`

Snap segments and vertices of input geometry to vertices of a reference geometry.

see [http://postgis.net/docs/ST\\_Snap.html](http://postgis.net/docs/ST_Snap.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SnapToGrid(*args, **kwargs)`

[geometry] Snap all points of the input geometry to a regular grid. OR [raster] Resample a raster by snapping it to a grid. New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline or Lanczos resampling algorithm. Default is NearestNeighbor.

see [http://postgis.net/docs/ST\\_SnapToGrid.html](http://postgis.net/docs/ST_SnapToGrid.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Split(*args, **kwargs)`

Returns a collection of geometries resulting by splitting a geometry.

see [http://postgis.net/docs/ST\\_Split.html](http://postgis.net/docs/ST_Split.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_StartPoint(*args, **kwargs)`

Returns the first point of a LineString.

see [http://postgis.net/docs/ST\\_StartPoint.html](http://postgis.net/docs/ST_StartPoint.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_StdDev4ma(*args, **kwargs)`

Raster processing function that calculates the standard deviation of pixel values in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_StdDev4ma.html](http://postgis.net/docs/RT_ST_StdDev4ma.html)

**class** `geoalchemy2.functions.ST_StraightSkeleton(*args, **kwargs)`

Compute a straight skeleton from a geometry

see [http://postgis.net/docs/ST\\_StraightSkeleton.html](http://postgis.net/docs/ST_StraightSkeleton.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Subdivide(*args, **kwargs)`

Returns a set of geometry where no geometry in the set has more than the specified number of vertices.

see [http://postgis.net/docs/ST\\_Subdivide.html](http://postgis.net/docs/ST_Subdivide.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Sum4ma(*args, **kwargs)`

Raster processing function that calculates the sum of all pixel values in a neighborhood.

see [http://postgis.net/docs/RT\\_ST\\_Sum4ma.html](http://postgis.net/docs/RT_ST_Sum4ma.html)

**class** `geoalchemy2.functions.ST_Summary(*args, **kwargs)`

[geometry] Returns a text summary of the contents of a geometry. OR [raster] Returns a text summary of the contents of the raster.

see [http://postgis.net/docs/ST\\_Summary.html](http://postgis.net/docs/ST_Summary.html)

**class** `geoalchemy2.functions.ST_SummaryStats(*args, **kwargs)`

Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a raster or raster coverage. Band 1 is assumed is no band is specified.

see [http://postgis.net/docs/RT\\_ST\\_SummaryStats.html](http://postgis.net/docs/RT_ST_SummaryStats.html)

**class** `geoalchemy2.functions.ST_SummaryStatsAgg(*args, **kwargs)`

Aggregate. Returns summarystats consisting of count, sum, mean, stddev, min, max for a given raster band of a set of raster. Band 1 is assumed is no band is specified.

see [http://postgis.net/docs/RT\\_ST\\_SummaryStatsAgg.html](http://postgis.net/docs/RT_ST_SummaryStatsAgg.html)

Return type: *geoalchemy2.types.SummaryStats*.

**type**

alias of *SummaryStats*

**class** `geoalchemy2.functions.ST_SwapOrdinates(*args, **kwargs)`

Returns a version of the given geometry with given ordinate values swapped.

see [http://postgis.net/docs/ST\\_SwapOrdinates.html](http://postgis.net/docs/ST_SwapOrdinates.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_SymDifference(*args, **kwargs)`

Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because  $ST\_SymDifference(A,B) = ST\_SymDifference(B,A)$ .

see [http://postgis.net/docs/ST\\_SymDifference.html](http://postgis.net/docs/ST_SymDifference.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_TPI(*args, **kwargs)`

Returns a raster with the calculated Topographic Position Index.

see [http://postgis.net/docs/RT\\_ST\\_TPI.html](http://postgis.net/docs/RT_ST_TPI.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_TRI(*args, **kwargs)`

Returns a raster with the calculated Terrain Ruggedness Index.

see [http://postgis.net/docs/RT\\_ST\\_TRI.html](http://postgis.net/docs/RT_ST_TRI.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_Tesselate(*args, **kwargs)`

Perform surface Tessellation of a polygon or polyhedralsurface and returns as a TIN or collection of TINS

see [http://postgis.net/docs/ST\\_Tesselate.html](http://postgis.net/docs/ST_Tesselate.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Tile(*args, **kwargs)`

Returns a set of rasters resulting from the split of the input raster based upon the desired dimensions of the output rasters.

see [http://postgis.net/docs/RT\\_ST\\_Tile.html](http://postgis.net/docs/RT_ST_Tile.html)

Return type: *geoalchemy2.types.Raster*.

**type**

alias of *Raster*

**class** `geoalchemy2.functions.ST_TileEnvelope(*args, **kwargs)`

Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

see [http://postgis.net/docs/ST\\_TileEnvelope.html](http://postgis.net/docs/ST_TileEnvelope.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Touches(*args, **kwargs)`

[geometry] Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.  
OR [raster] Return true if raster rastA and rastB have at least one point in common but their interiors do not intersect.

see [http://postgis.net/docs/ST\\_Touches.html](http://postgis.net/docs/ST_Touches.html)

**class** `geoalchemy2.functions.ST_TransScale(*args, **kwargs)`

Translates and scales a geometry by given offsets and factors.

see [http://postgis.net/docs/ST\\_TransScale.html](http://postgis.net/docs/ST_TransScale.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Transform(*args, **kwargs)`

[geometry] Return a new geometry with its coordinates transformed to a different spatial reference system. OR [raster] Reprojects a raster in a known spatial reference system to another known spatial reference system using specified resampling algorithm. Options are NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos defaulting to NearestNeighbor.

see [http://postgis.net/docs/ST\\_Transform.html](http://postgis.net/docs/ST_Transform.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_Translate(*args, **kwargs)`

Translates a geometry by given offsets.

see [http://postgis.net/docs/ST\\_Translate.html](http://postgis.net/docs/ST_Translate.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** `geoalchemy2.functions.ST_UnaryUnion(*args, **kwargs)`

Like `ST_Union`, but working at the geometry component level.

see [http://postgis.net/docs/ST\\_UnaryUnion.html](http://postgis.net/docs/ST_UnaryUnion.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_Union(*args, **kwargs)`

[geometry] Returns a geometry that represents the point set union of the Geometries. OR [raster] Returns the union of a set of raster tiles into a single raster composed of 1 or more bands.

see [http://postgis.net/docs/ST\\_Union.html](http://postgis.net/docs/ST_Union.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** `geoalchemy2.functions.ST_UpperLeftX(*args, **kwargs)`

Returns the upper left X coordinate of raster in projected spatial ref.

see [http://postgis.net/docs/RT\\_ST\\_UpperLeftX.html](http://postgis.net/docs/RT_ST_UpperLeftX.html)

**class** `geoalchemy2.functions.ST_UpperLeftY(*args, **kwargs)`

Returns the upper left Y coordinate of raster in projected spatial ref.

see [http://postgis.net/docs/RT\\_ST\\_UpperLeftY.html](http://postgis.net/docs/RT_ST_UpperLeftY.html)

**class** `geoalchemy2.functions.ST_Value(*args, **kwargs)`

Returns the value of a given band in a given columnx, rowy pixel or at a particular geometric point. Band numbers start at 1 and assumed to be 1 if not specified. If `exclude_nodata_value` is set to false, then all pixels include nodata pixels are considered to intersect and return value. If `exclude_nodata_value` is not passed in then reads it from metadata of raster.

see [http://postgis.net/docs/RT\\_ST\\_Value.html](http://postgis.net/docs/RT_ST_Value.html)

**class** `geoalchemy2.functions.ST_ValueCount(*args, **kwargs)`

Returns a set of records containing a pixel band value and count of the number of pixels in a given band of a raster (or a raster coverage) that have a given set of values. If no band is specified defaults to band 1. By default nodata value pixels are not counted. and all other values in the pixel are output and pixel band values are rounded to the nearest integer.

see [http://postgis.net/docs/RT\\_ST\\_ValueCount.html](http://postgis.net/docs/RT_ST_ValueCount.html)

**class** `geoalchemy2.functions.ST_Volume(*args, **kwargs)`

Computes the volume of a 3D solid. If applied to surface (even closed) geometries will return 0.

see [http://postgis.net/docs/ST\\_Volume.html](http://postgis.net/docs/ST_Volume.html)

**class** `geoalchemy2.functions.ST_VoronoiLines(*args, **kwargs)`

Returns the boundaries between the cells of the Voronoi diagram constructed from the vertices of a geometry.

see [http://postgis.net/docs/ST\\_VoronoiLines.html](http://postgis.net/docs/ST_VoronoiLines.html)

Return type: `geoalchemy2.types.Geometry`.

**type**

alias of `Geometry`

**class** geoalchemy2.functions.**ST\_VoronoiPolygons**(\*args, \*\*kwargs)

Returns the cells of the Voronoi diagram constructed from the vertices of a geometry.

see [http://postgis.net/docs/ST\\_VoronoiPolygons.html](http://postgis.net/docs/ST_VoronoiPolygons.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_WKBToSQL**(\*args, \*\*kwargs)

Return a specified ST\_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST\_GeomFromWKB that takes no srid

see [http://postgis.net/docs/ST\\_WKBToSQL.html](http://postgis.net/docs/ST_WKBToSQL.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_WKTToSQL**(\*args, \*\*kwargs)

Return a specified ST\_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST\_GeomFromText

see [http://postgis.net/docs/ST\\_WKTToSQL.html](http://postgis.net/docs/ST_WKTToSQL.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_Width**(\*args, \*\*kwargs)

Returns the width of the raster in pixels.

see [http://postgis.net/docs/RT\\_ST\\_Width.html](http://postgis.net/docs/RT_ST_Width.html)

**class** geoalchemy2.functions.**ST\_Within**(\*args, \*\*kwargs)

[geometry] Returns true if the geometry A is completely inside geometry B OR [raster] Return true if no points of raster rastA lie in the exterior of raster rastB and at least one point of the interior of rastA lies in the interior of rastB.

see [http://postgis.net/docs/ST\\_Within.html](http://postgis.net/docs/ST_Within.html)

**class** geoalchemy2.functions.**ST\_WorldToRasterCoord**(\*args, \*\*kwargs)

Returns the upper left corner as column and row given geometric X and Y (longitude and latitude) or a point geometry expressed in the spatial reference coordinate system of the raster.

see [http://postgis.net/docs/RT\\_ST\\_WorldToRasterCoord.html](http://postgis.net/docs/RT_ST_WorldToRasterCoord.html)

**class** geoalchemy2.functions.**ST\_WorldToRasterCoordX**(\*args, \*\*kwargs)

Returns the column in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

see [http://postgis.net/docs/RT\\_ST\\_WorldToRasterCoordX.html](http://postgis.net/docs/RT_ST_WorldToRasterCoordX.html)

**class** geoalchemy2.functions.**ST\_WorldToRasterCoordY**(\*args, \*\*kwargs)

Returns the row in the raster of the point geometry (pt) or a X and Y world coordinate (xw, yw) represented in world spatial reference system of raster.

see [http://postgis.net/docs/RT\\_ST\\_WorldToRasterCoordY.html](http://postgis.net/docs/RT_ST_WorldToRasterCoordY.html)



**class** geoalchemy2.functions.**ST\_WrapX**(\*args, \*\*kwargs)

Wrap a geometry around an X value.

see [http://postgis.net/docs/ST\\_WrapX.html](http://postgis.net/docs/ST_WrapX.html)

Return type: *geoalchemy2.types.Geometry*.

**type**

alias of *Geometry*

**class** geoalchemy2.functions.**ST\_X**(\*args, \*\*kwargs)

Returns the X coordinate of a Point.

see [http://postgis.net/docs/ST\\_X.html](http://postgis.net/docs/ST_X.html)

**class** geoalchemy2.functions.**ST\_XMax**(\*args, \*\*kwargs)

Returns the X maxima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_XMax.html](http://postgis.net/docs/ST_XMax.html)

**class** geoalchemy2.functions.**ST\_XMin**(\*args, \*\*kwargs)

Returns the X minima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_XMin.html](http://postgis.net/docs/ST_XMin.html)

**class** geoalchemy2.functions.**ST\_Y**(\*args, \*\*kwargs)

Returns the Y coordinate of a Point.

see [http://postgis.net/docs/ST\\_Y.html](http://postgis.net/docs/ST_Y.html)

**class** geoalchemy2.functions.**ST\_YMax**(\*args, \*\*kwargs)

Returns the Y maxima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_YMax.html](http://postgis.net/docs/ST_YMax.html)

**class** geoalchemy2.functions.**ST\_YMin**(\*args, \*\*kwargs)

Returns the Y minima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_YMin.html](http://postgis.net/docs/ST_YMin.html)

**class** geoalchemy2.functions.**ST\_Z**(\*args, \*\*kwargs)

Returns the Z coordinate of a Point.

see [http://postgis.net/docs/ST\\_Z.html](http://postgis.net/docs/ST_Z.html)

**class** geoalchemy2.functions.**ST\_ZMax**(\*args, \*\*kwargs)

Returns the Z maxima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_ZMax.html](http://postgis.net/docs/ST_ZMax.html)

**class** geoalchemy2.functions.**ST\_ZMin**(\*args, \*\*kwargs)

Returns the Z minima of a 2D or 3D bounding box or a geometry.

see [http://postgis.net/docs/ST\\_ZMin.html](http://postgis.net/docs/ST_ZMin.html)

**class** geoalchemy2.functions.**ST\_Zmflag**(\*args, \*\*kwargs)

Returns a code indicating the ZM coordinate dimension of a geometry.

see [http://postgis.net/docs/ST\\_Zmflag.html](http://postgis.net/docs/ST_Zmflag.html)

**class** geoalchemy2.functions.**TableRowElement**(selectable)

**class** geoalchemy2.functions.UnlockRows(\*args, \*\*kwargs)

Removes all locks held by an authorization token.

see <http://postgis.net/docs/UnlockRows.html>

**class** geoalchemy2.functions.UpdateGeometrySRID(\*args, \*\*kwargs)

Updates the SRID of all features in a geometry column, and the table metadata.

see <http://postgis.net/docs/UpdateGeometrySRID.html>

**class** geoalchemy2.functions.postgis\_sfcgal\_version(\*args, \*\*kwargs)

Returns the version of SFCGAL in use

see [http://postgis.net/docs/postgis\\_sfcgal\\_version.html](http://postgis.net/docs/postgis_sfcgal_version.html)

## 7.4 Spatial Operators

This module defines a `Comparator` class for use with geometry and geography objects.

This is where spatial operators, like `&&` and `<<`, are defined. Spatial operators very often apply to the bounding boxes of geometries. For example, `geom1 && geom2` indicates if `geom1`'s bounding box intersects `geom2`'s.

### Examples

Select the objects whose bounding boxes are to the left of the bounding box of `POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))`:

```
select([table]).where(table.c.geom.to_left(
    'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))'))
```

The `<<` and `>>` operators are a bit specific, because they have corresponding Python operator (`__lshift__` and `__rshift__`). The above `SELECT` expression can thus be rewritten like this:

```
select([table]).where(
    table.c.geom << 'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))')
```

Operators can also be used when using the ORM. For example:

```
Session.query(Cls).filter(
    Cls.geom << 'POLYGON((-5 45,5 45,5 -45,-5 -45,-5 45))')
```

Now some other examples with the `<#>` operator.

Select the ten objects that are the closest to `POINT(0 0)` (typical closed neighbors problem):

```
select([table]).order_by(table.c.geom.distance_box('POINT(0 0)')).limit(10)
```

Using the ORM:

```
Session.query(Cls).order_by(Cls.geom.distance_box('POINT(0 0)')).limit(10)
```

**class** geoalchemy2.comparator.BaseComparator(expr: ColumnElement[\_CT])

Bases: `Comparator`

A custom comparator base class.

It adds the ability to call spatial functions on columns that use this kind of comparator. It also defines functions that map to operators supported by `Geometry`, `Geography` and `Raster` columns.

This comparator is used by the `geoalchemy2.types.Raster`.

#### `__weakref__`

list of weak references to the object (if defined)

#### `intersects(other)`

The `&&` operator. A's BBOX intersects B's.

#### `overlaps_or_to_left(other)`

The `&<` operator. A's BBOX overlaps or is to the left of B's.

#### `overlaps_or_to_right(other)`

The `&>` operator. A's BBOX overlaps or is to the right of B's.

**class** `geoalchemy2.comparator.Comparator`(*expr: ColumnElement[\_CT]*)

Bases: `BaseComparator`

A custom comparator class.

Used in `geoalchemy2.types.Geometry` and `geoalchemy2.types.Geography`.

This is where spatial operators like `<<` and `<->` are defined.

#### `__lshift__(other)`

The `<<` operator.

A's BBOX is strictly to the left of B's.

Same as `to_left`, so:

```
table.c.geom << 'POINT(1 2)'
```

is the same as:

```
table.c.geom.to_left('POINT(1 2)')
```

#### `__rshift__(other)`

The `>>` operator.

A's BBOX is strictly to the left of B's.

Same as `to_`right``, so:

```
table.c.geom >> 'POINT(1 2)'
```

is the same as:

```
table.c.geom.to_right('POINT(1 2)')
```

#### `above(other)`

The `|>>` operator.

A's BBOX is strictly above B's.

#### `below(other)`

The `<<|` operator.

A's BBOX is strictly below B's.

**contained**(*other*)

The @ operator.

A's BBOX is contained by B's.

**contains**(*other*, *\*\*kw*)

The ~ operator.

A's BBOX contains B's.

**distance\_box**(*other*)

The <#> operator.

The distance between bounding box of two geometries.

**distance\_centroid**(*other*)

The <-> operator.

The distance between two points.

**intersects\_nd**(*other*)

The &&& operator.

This operator returns TRUE if the n-D bounding box of geometry A intersects the n-D bounding box of geometry B.

**overlaps\_or\_above**(*other*)

The |&> operator.

A's BBOX overlaps or is above B's.

**overlaps\_or\_below**(*other*)

The &<| operator.

A's BBOX overlaps or is below B's.

**same**(*other*)

The ~= operator.

A's BBOX is the same as B's.

**to\_left**(*other*)

The << operator.

A's BBOX is strictly to the left of B's.

**to\_right**(*other*)

The >> operator.

A's BBOX is strictly to the right of B's.

## 7.5 Shapely Integration

This module provides utility functions for integrating with Shapely.

**Note:** As GeoAlchemy 2 itself has no dependency on *Shapely*, applications using functions of this module have to ensure that *Shapely* is available.

`geoalchemy2.shape.from_shape(shape, srid=-1, extended=False)`

Function to convert a Shapely geometry to a `geoalchemy2.types.WKBElement`.

### Parameters

- **shape** – The shape to convert.
- **srid** – An integer representing the spatial reference system. E.g. 4326. Default value is -1, which means no/unknown reference system.
- **extended** – A boolean to switch between WKB and EWKB. Default value is False.

Example:

```
from shapely.geometry import Point
wkb_element = from_shape(Point(5, 45), srid=4326)
ewkb_element = from_shape(Point(5, 45), srid=4326, extended=True)
```

`geoalchemy2.shape.to_shape(element)`

Function to convert a `geoalchemy2.types.SpatialElement` to a Shapely geometry.

### Parameters

- **element** – The element to convert into a Shapely object.

Example:

```
lake = Session.query(Lake).get(1)
polygon = to_shape(lake.geom)
```

## 7.6 Alembic helpers

Some helpers to use with Alembic migration tool.

**class** `geoalchemy2.alembic_helpers.AddGeospatialColumnOp`(*table\_name: str, column: Column[Any], \*, schema: Optional[str] = None, \*\*kw: Any*)

Bases: `AddColumnOp`

Add a Geospatial Column in an Alembic migration context.

This method originates from: <https://alembic.sqlalchemy.org/en/latest/api/operations.html#operation-plugins>

**classmethod** `add_geospatial_column`(*operations, table\_name, column, schema=None*)

This method is proxied on the `Operations` class, via the `Operations.add_geospatial_column()` method.

**classmethod** `batch_add_geospatial_column`(*operations, column, insert\_before=None, insert\_after=None*)

This method is proxied on the `BatchOperations` class, via the `BatchOperations.add_geospatial_column()` method.

**reverse()**

Used to autogenerate the downgrade function.

```
class geoalchemy2.alembic_helpers.CreateGeospatialIndexOp(index_name: Optional[str], table_name: str, columns: Sequence[Union[str, TextClause, ColumnElement[Any]]], *, schema: Optional[str] = None, unique: bool = False, **kw: Any)
```

Bases: `CreateIndexOp`

```
classmethod batch_create_geospatial_index(operations, index_name, columns, **kw)
```

This method is proxied on the `BatchOperations` class, via the `BatchOperations.create_geospatial_index()` method.

```
classmethod create_geospatial_index(operations, index_name, table_name, columns, schema=None, unique=False, **kw)
```

This method is proxied on the `Operations` class, via the `Operations.create_geospatial_index()` method.

**reverse()**

Used to autogenerate the downgrade function.

```
class geoalchemy2.alembic_helpers.CreateGeospatialTableOp(table_name: str, columns: Sequence[SchemaItem], *, schema: Optional[str] = None, _namespace_metadata: Optional[MetaData] = None, _constraints_included: bool = False, **kw: Any)
```

Bases: `CreateTableOp`

Create a Geospatial Table in an Alembic migration context.

This method originates from: <https://alembic.sqlalchemy.org/en/latest/api/operations.html#operation-plugins>

```
classmethod create_geospatial_table(operations, table_name, *columns, **kw)
```

This method is proxied on the `Operations` class, via the `Operations.create_geospatial_table()` method.

**reverse()**

Used to autogenerate the downgrade function.

```
class geoalchemy2.alembic_helpers.DropGeospatialColumnOp(table_name: str, column_name: str, *, schema: Optional[str] = None, _reverse: Optional[AddColumnOp] = None, **kw: Any)
```

Bases: `DropColumnOp`

Drop a Geospatial Column in an Alembic migration context.

```
classmethod batch_drop_geospatial_column(operations, column_name, **kw)
```

This method is proxied on the `BatchOperations` class, via the `BatchOperations.drop_geospatial_column()` method.

**classmethod** `drop_geospatial_column`(*operations*, *table\_name*, *column\_name*, *schema=None*, *\*\*kw*)

This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_column()` method.

**reverse()**

Used to autogenerate the downgrade function.

**class** `geoalchemy2.alembic_helpers.DropGeospatialIndexOp`(\**args*, *column\_name*, *\*\*kwargs*)

Bases: `DropIndexOp`

**classmethod** `batch_drop_geospatial_index`(*operations*, *index\_name*, *\*\*kw*)

This method is proxied on the `BatchOperations` class, via the `BatchOperations.drop_geospatial_index()` method.

**classmethod** `drop_geospatial_index`(*operations*, *index\_name*, *table\_name*, *column\_name*, *schema=None*, *unique=False*, *\*\*kw*)

This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_index()` method.

**reverse()**

Used to autogenerate the downgrade function.

**class** `geoalchemy2.alembic_helpers.DropGeospatialTableOp`(*table\_name: str*, \*, *schema: Optional[str] = None*, *table\_kw: Optional[MutableMapping[Any, Any]] = None*, *\_reverse: Optional[CreateTableOp] = None*)

Bases: `DropTableOp`

**classmethod** `drop_geospatial_table`(*operations*, *table\_name*, *schema=None*, *\*\*kw*)

This method is proxied on the `Operations` class, via the `Operations.drop_geospatial_table()` method.

**reverse()**

Used to autogenerate the downgrade function.

`geoalchemy2.alembic_helpers.add_geo_column`(*context*, *revision*, *op*)

Replace the default `AddColumnOp` by a geospatial-specific one.

`geoalchemy2.alembic_helpers.add_geospatial_column`(*operations*, *operation*)

Handle the actual column addition according to the dialect backend.

#### Parameters

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – `AddGeospatialColumnOp` call, with attributes for `table_name`, `column_name`, `column_type`, and optional keywords.

`geoalchemy2.alembic_helpers.create_geo_index`(*context*, *revision*, *op*)

Replace the default `CreateIndexOp` by a geospatial-specific one.

`geoalchemy2.alembic_helpers.create_geo_table`(*context*, *revision*, *op*)

Replace the default `CreateTableOp` by a geospatial-specific one.

`geoalchemy2.alembic_helpers.create_geospatial_index(operations, operation)`

Handle the actual index creation according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – CreateGeospatialIndexOp call, with attributes for table\_name, column\_name, column\_type, and optional keywords.

`geoalchemy2.alembic_helpers.create_geospatial_table(operations, operation)`

Handle the actual table creation according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – CreateGeospatialTableOp call, with attributes for table\_name, column\_name, column\_type, and optional keywords.

`geoalchemy2.alembic_helpers.drop_geo_column(context, revision, op)`

Replace the default DropColumnOp by a geospatial-specific one.

`geoalchemy2.alembic_helpers.drop_geo_index(context, revision, op)`

Replace the default DropIndexOp by a geospatial-specific one.

`geoalchemy2.alembic_helpers.drop_geo_table(context, revision, op)`

Replace the default DropTableOp by a geospatial-specific one.

`geoalchemy2.alembic_helpers.drop_geospatial_column(operations, operation)`

Handle the actual column removal according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – AddGeospatialColumnOp call, with attributes for table\_name, column\_name, column\_type, and optional keywords.

`geoalchemy2.alembic_helpers.drop_geospatial_index(operations, operation)`

Handle the actual index drop according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – DropGeospatialIndexOp call, with attributes for table\_name, column\_name, column\_type, and optional keywords.

`geoalchemy2.alembic_helpers.drop_geospatial_table(operations, operation)`

Handle the actual table removal according to the dialect backend.

**Parameters**

- **operations** – Operations object from alembic base, defining high level migration operations.
- **operation** – DropGeospatialTableOp call, with attributes for table\_name, column\_name, column\_type, and optional keywords.



`geoalchemy2.alembic_helpers.include_object(obj, name, obj_type, reflected, compare_to)`

Do not include internal tables of spatial extensions.

**Warning:** This function only checks the table names, so it might exclude tables that should not be. In such case, you should create your own function to handle your specific table names.

`geoalchemy2.alembic_helpers.render_add_geo_column(autogen_context, op)`

Render the `add_geospatial_column` operation in migration script.

`geoalchemy2.alembic_helpers.render_create_geo_index(autogen_context, op)`

Render the `create_geospatial_index` operation in migration script.

`geoalchemy2.alembic_helpers.render_create_geo_table(autogen_context, op)`

Render the `create_geospatial_table` operation in migration script.

`geoalchemy2.alembic_helpers.render_drop_geo_column(autogen_context, op)`

Render the `drop_geospatial_column` operation in migration script.

`geoalchemy2.alembic_helpers.render_drop_geo_index(autogen_context, op)`

Render the `drop_geospatial_index` operation in migration script.

`geoalchemy2.alembic_helpers.render_drop_geo_table(autogen_context, op)`

Render the `drop_geospatial_table` operation in migration script.

`geoalchemy2.alembic_helpers.render_item(obj_type, obj, autogen_context)`

Add proper imports for spatial types.

`geoalchemy2.alembic_helpers.visit_drop_geospatial_table(element, compiler, **kw)`

Specific compilation rule to drop spatial tables with SQLite dialect.

`geoalchemy2.alembic_helpers.visit_rename_geospatial_table(element, compiler, **kw)`

Specific compilation rule to rename spatial tables with SQLite dialect.

`geoalchemy2.alembic_helpers.writer = <alembic.autogenerate.rewriter.Rewriter object>`

Rewriter object for Alembic.



## DEVELOPMENT

The code is available on GitHub: <https://github.com/geoalchemy/geoalchemy2>.

Contributors:

- Adrien Berchet (<https://github.com/adrien-berchet>)
- Éric Lemoine (<https://github.com/elemoine>)
- Dolf Andringa (<https://github.com/dolfandringa>)
- Frédéric Junod, Camptocamp SA (<https://github.com/fredj>)
- ijl (<https://github.com/ijl>)
- Loïc Gasser (<https://github.com/loicgasser>)
- Marcel Radischat (<https://github.com/quiqua>)
- rapto (<https://github.com/rapto>)
- Serge Bouchut (<https://github.com/SergeBouchut>)
- Tobias Bieniek (<https://github.com/Turbo87>)
- Tom Payne (<https://github.com/twpayne>)

Many thanks to Mike Bayer for his guidance and support! He also [fostered](#) the birth of GeoAlchemy 2.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### g

- `geoalchemy2.alembic_helpers`, [113](#)
- `geoalchemy2.comparator`, [110](#)
- `geoalchemy2.functions`, [57](#)
- `geoalchemy2.shape`, [113](#)
- `geoalchemy2.types`, [51](#)





## Symbols

`_DummyGeometry` (class in `geoalchemy2.types`), 53  
`_GISType` (class in `geoalchemy2.types`), 53  
`_SpatialElement` (class in `geoalchemy2.elements`), 55  
`__lshift__()` (`geoalchemy2.comparator.Comparator` method), 111  
`__rshift__()` (`geoalchemy2.comparator.Comparator` method), 111  
`__weakref__` (`geoalchemy2.comparator.BaseComparator` attribute), 111

## A

`above()` (`geoalchemy2.comparator.Comparator` method), 111  
`add_geo_column()` (in module `geoalchemy2.alembic_helpers`), 115  
`add_geospatial_column()` (`geoalchemy2.alembic_helpers.AddGeospatialColumnOp` class method), 113  
`add_geospatial_column()` (in module `geoalchemy2.alembic_helpers`), 115  
`AddAuth` (class in `geoalchemy2.functions`), 58  
`AddGeometryColumn` (class in `geoalchemy2.functions`), 58  
`AddGeospatialColumnOp` (class in `geoalchemy2.alembic_helpers`), 113  
`as_binary` (`geoalchemy2.types._GISType` attribute), 55  
`as_binary` (`geoalchemy2.types.Geography` attribute), 51  
`as_binary` (`geoalchemy2.types.Geometry` attribute), 52  
`as_binary` (`geoalchemy2.types.Raster` attribute), 53

## B

`BaseComparator` (class in `geoalchemy2.comparator`), 110  
`batch_add_geospatial_column()` (`geoalchemy2.alembic_helpers.AddGeospatialColumnOp` class method), 113  
`batch_create_geospatial_index()` (`geoalchemy2.alembic_helpers.CreateGeospatialIndexOp` class method), 114  
`batch_drop_geospatial_column()` (`geoalchemy2.alembic_helpers.DropGeospatialColumnOp`

class method), 114  
`batch_drop_geospatial_index()` (`geoalchemy2.alembic_helpers.DropGeospatialIndexOp` class method), 115  
`below()` (`geoalchemy2.comparator.Comparator` method), 111  
`bind_expression()` (`geoalchemy2.types._GISType` method), 55  
`bind_processor()` (`geoalchemy2.types._GISType` method), 55  
`Box2D` (class in `geoalchemy2.functions`), 58  
`Box3D` (class in `geoalchemy2.functions`), 58

## C

`cache_ok` (`geoalchemy2.types._GISType` attribute), 55  
`cache_ok` (`geoalchemy2.types.Geography` attribute), 51  
`cache_ok` (`geoalchemy2.types.Geometry` attribute), 52  
`cache_ok` (`geoalchemy2.types.GeometryDump` attribute), 52  
`cache_ok` (`geoalchemy2.types.Raster` attribute), 53  
`cache_ok` (`geoalchemy2.types.SummaryStats` attribute), 53  
`CheckAuth` (class in `geoalchemy2.functions`), 58  
`column_expression()` (`geoalchemy2.types._GISType` method), 55  
`Comparator` (class in `geoalchemy2.comparator`), 111  
`comparator_factory` (`geoalchemy2.types._GISType` attribute), 55  
`comparator_factory` (`geoalchemy2.types.Raster` attribute), 53  
`CompositeElement` (class in `geoalchemy2.elements`), 56  
`CompositeType` (class in `geoalchemy2.types`), 51  
`CompositeType.comparator_factory` (class in `geoalchemy2.types`), 51  
`contained()` (`geoalchemy2.comparator.Comparator` method), 111  
`contains()` (`geoalchemy2.comparator.Comparator` method), 112  
`create_geo_index()` (in module `geoalchemy2.alembic_helpers`), 115  
`create_geo_table()` (in module `geoalchemy2.alembic_helpers`), 115

[create\\_geospatial\\_index\(\)](#) ([geoalchemy2.alembic\\_helpers.CreateGeospatialIndexOp](#) [geoalchemy2.alembic\\_helpers](#)), 115  
 class method), 114

[create\\_geospatial\\_index\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 115

[create\\_geospatial\\_table\(\)](#) ([geoalchemy2.alembic\\_helpers.CreateGeospatialTableOp](#) [geoalchemy2.alembic\\_helpers](#)), 115  
 class method), 114

[create\\_geospatial\\_table\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[CreateGeospatialIndexOp](#) (class in [geoalchemy2.alembic\\_helpers](#)), 114

[CreateGeospatialTableOp](#) (class in [geoalchemy2.alembic\\_helpers](#)), 114

## D

[desc](#) ([geoalchemy2.elements.RasterElement](#) property), 56

[desc](#) ([geoalchemy2.elements.WKBElement](#) property), 56

[desc](#) ([geoalchemy2.elements.WKTElement](#) property), 56

[DisableLongTransactions](#) (class in [geoalchemy2.functions](#)), 58

[distance\\_box\(\)](#) ([geoalchemy2.comparator.Comparator](#) method), 112

[distance\\_centroid\(\)](#) ([geoalchemy2.comparator.Comparator](#) method), 112

[drop\\_geo\\_column\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[drop\\_geo\\_index\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[drop\\_geo\\_table\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[drop\\_geospatial\\_column\(\)](#) ([geoalchemy2.alembic\\_helpers.DropGeospatialColumnOp](#) [geoalchemy2.alembic\\_helpers](#)), 115  
 class method), 114

[drop\\_geospatial\\_column\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[drop\\_geospatial\\_index\(\)](#) ([geoalchemy2.alembic\\_helpers.DropGeospatialIndexOp](#) [geoalchemy2.alembic\\_helpers](#)), 115  
 class method), 115

[drop\\_geospatial\\_index\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[drop\\_geospatial\\_table\(\)](#) ([geoalchemy2.alembic\\_helpers.DropGeospatialTableOp](#) [geoalchemy2.alembic\\_helpers](#)), 115  
 class method), 115

[drop\\_geospatial\\_table\(\)](#) (in module [geoalchemy2.alembic\\_helpers](#)), 116

[DropGeometryColumn](#) (class in [geoalchemy2.functions](#)), 58

[DropGeometryTable](#) (class in [geoalchemy2.functions](#)), 58

[DropGeospatialColumnOp](#) (class in [geoalchemy2.alembic\\_helpers](#)), 114

[DropGeospatialIndexOp](#) (class in [geoalchemy2.alembic\\_helpers](#)), 115

[DropGeospatialTableOp](#) (class in [geoalchemy2.alembic\\_helpers](#)), 115

## E

[ElementType](#) ([geoalchemy2.types.Geography](#) attribute), 51

[ElementType](#) ([geoalchemy2.types.Geometry](#) attribute), 52

[ElementType](#) ([geoalchemy2.types.Raster](#) attribute), 53

[EnableLongTransactions](#) (class in [geoalchemy2.functions](#)), 58

## F

[Find\\_SRID](#) (class in [geoalchemy2.functions](#)), 58

[from\\_shape\(\)](#) (in module [geoalchemy2.shape](#)), 113

[from\\_text](#) ([geoalchemy2.types.\\_GISType](#) attribute), 55

[from\\_text](#) ([geoalchemy2.types.Geography](#) attribute), 52

[from\\_text](#) ([geoalchemy2.types.Geometry](#) attribute), 52

[from\\_text](#) ([geoalchemy2.types.Raster](#) attribute), 53

## G

[GenericFunction](#) (class in [geoalchemy2.functions](#)), 58

[geoalchemy2.alembic\\_helpers](#) module, 113

[geoalchemy2.comparator](#) module, 110

[geoalchemy2.functions](#) module, 57

[geoalchemy2.shape](#) module, 113

[geoalchemy2.types](#) module, 51

[GeoGenericFunction](#) (class in [geoalchemy2.functions](#)), 59

[Geography](#) (class in [geoalchemy2.types](#)), 51

[geom\\_from](#) ([geoalchemy2.elements.WKBElement](#) attribute), 56

[geom\\_from](#) ([geoalchemy2.elements.WKTElement](#) attribute), 56

[geom\\_from\\_extended\\_version](#) ([geoalchemy2.elements.WKBElement](#) attribute), 56

[geom\\_from\\_extended\\_version](#) ([geoalchemy2.elements.WKTElement](#) attribute), 56

[Geometry](#) (class in [geoalchemy2.types](#)), 52

[GeometryDump](#) (class in [geoalchemy2.types](#)), 52

[GeometryType](#) (class in [geoalchemy2.functions](#)), 59

## H

[HasFunction](#) (class in [geoalchemy2.elements](#)), 55

## I

`include_object()` (in module `geoalchemy2.alembic_helpers`), 116

`intersects()` (`geoalchemy2.comparator.BaseComparator` method), 111

`intersects_nd()` (`geoalchemy2.comparator.Comparator` method), 112

## L

`LockRow` (class in `geoalchemy2.functions`), 59

## M

module

- `geoalchemy2.alembic_helpers`, 113
- `geoalchemy2.comparator`, 110
- `geoalchemy2.functions`, 57
- `geoalchemy2.shape`, 113
- `geoalchemy2.types`, 51

## N

`name` (`geoalchemy2.types._GISType` attribute), 55

`name` (`geoalchemy2.types.Geography` attribute), 52

`name` (`geoalchemy2.types.Geometry` attribute), 52

`name` (`geoalchemy2.types.Raster` attribute), 53

## O

`overlaps_or_above()` (`geoalchemy2.comparator.Comparator` method), 112

`overlaps_or_below()` (`geoalchemy2.comparator.Comparator` method), 112

`overlaps_or_to_left()` (`geoalchemy2.comparator.BaseComparator` method), 111

`overlaps_or_to_right()` (`geoalchemy2.comparator.BaseComparator` method), 111

## P

`Populate_Geometry_Columns` (class in `geoalchemy2.functions`), 59

`PostGIS_AddBBBox` (class in `geoalchemy2.functions`), 59

`PostGIS_DropBBBox` (class in `geoalchemy2.functions`), 59

`PostGIS_Extensions_Upgrade` (class in `geoalchemy2.functions`), 59

`PostGIS_Full_Version` (class in `geoalchemy2.functions`), 59

`PostGIS_GEOS_Version` (class in `geoalchemy2.functions`), 59

`PostGIS_HasBBBox` (class in `geoalchemy2.functions`), 60

`PostGIS_Lib_Build_Date` (class in `geoalchemy2.functions`), 60

`PostGIS_Lib_Version` (class in `geoalchemy2.functions`), 60

`PostGIS_Liblwgeom_Version` (class in `geoalchemy2.functions`), 60

`PostGIS_LibXML_Version` (class in `geoalchemy2.functions`), 60

`PostGIS_PROJ_Version` (class in `geoalchemy2.functions`), 60

`PostGIS_Scripts_Build_Date` (class in `geoalchemy2.functions`), 60

`PostGIS_Scripts_Installed` (class in `geoalchemy2.functions`), 60

`PostGIS_Scripts_Released` (class in `geoalchemy2.functions`), 60

`postgis_sfcgal_version` (class in `geoalchemy2.functions`), 110

`PostGIS_Version` (class in `geoalchemy2.functions`), 60

`PostGIS_Wagyu_Version` (class in `geoalchemy2.functions`), 60

## R

`Raster` (class in `geoalchemy2.types`), 52

`RasterElement` (class in `geoalchemy2.elements`), 56

`render_add_geo_column()` (in module `geoalchemy2.alembic_helpers`), 117

`render_create_geo_index()` (in module `geoalchemy2.alembic_helpers`), 117

`render_create_geo_table()` (in module `geoalchemy2.alembic_helpers`), 117

`render_drop_geo_column()` (in module `geoalchemy2.alembic_helpers`), 117

`render_drop_geo_index()` (in module `geoalchemy2.alembic_helpers`), 117

`render_drop_geo_table()` (in module `geoalchemy2.alembic_helpers`), 117

`render_item()` (in module `geoalchemy2.alembic_helpers`), 117

`result_processor()` (`geoalchemy2.types._GISType` method), 55

`reverse()` (`geoalchemy2.alembic_helpers.AddGeospatialColumnOp` method), 114

`reverse()` (`geoalchemy2.alembic_helpers.CreateGeospatialIndexOp` method), 114

`reverse()` (`geoalchemy2.alembic_helpers.CreateGeospatialTableOp` method), 114

`reverse()` (`geoalchemy2.alembic_helpers.DropGeospatialColumnOp` method), 115

`reverse()` (`geoalchemy2.alembic_helpers.DropGeospatialIndexOp` method), 115

`reverse()` (`geoalchemy2.alembic_helpers.DropGeospatialTableOp` method), 115

## S

`same()` (`geoalchemy2.comparator.Comparator` method),

- 112
- ST\_3DArea (class in *geoalchemy2.functions*), 60
- ST\_3DClosestPoint (class in *geoalchemy2.functions*), 61
- ST\_3DDFullyWithin (class in *geoalchemy2.functions*), 61
- ST\_3DDifference (class in *geoalchemy2.functions*), 61
- ST\_3DDistance (class in *geoalchemy2.functions*), 61
- ST\_3DDWithin (class in *geoalchemy2.functions*), 61
- ST\_3DExtent (class in *geoalchemy2.functions*), 61
- ST\_3DIntersection (class in *geoalchemy2.functions*), 61
- ST\_3DIntersects (class in *geoalchemy2.functions*), 61
- ST\_3DLength (class in *geoalchemy2.functions*), 61
- ST\_3DLineInterpolatePoint (class in *geoalchemy2.functions*), 62
- ST\_3DLongestLine (class in *geoalchemy2.functions*), 62
- ST\_3DMakeBox (class in *geoalchemy2.functions*), 62
- ST\_3DMaxDistance (class in *geoalchemy2.functions*), 62
- ST\_3DPerimeter (class in *geoalchemy2.functions*), 62
- ST\_3DShortestLine (class in *geoalchemy2.functions*), 62
- ST\_3DUnion (class in *geoalchemy2.functions*), 62
- ST\_AddBand (class in *geoalchemy2.functions*), 62
- ST\_AddMeasure (class in *geoalchemy2.functions*), 63
- ST\_AddPoint (class in *geoalchemy2.functions*), 63
- ST\_Affine (class in *geoalchemy2.functions*), 63
- ST\_Angle (class in *geoalchemy2.functions*), 63
- ST\_ApproximateMedialAxis (class in *geoalchemy2.functions*), 63
- ST\_Area (class in *geoalchemy2.functions*), 63
- ST\_AsBinary (class in *geoalchemy2.functions*), 63
- ST\_AsEncodedPolyline (class in *geoalchemy2.functions*), 64
- ST\_AsEWKB (class in *geoalchemy2.functions*), 64
- ST\_AsEWKT (class in *geoalchemy2.functions*), 64
- ST\_AsGDALRaster (class in *geoalchemy2.functions*), 64
- ST\_AsGeobuf (class in *geoalchemy2.functions*), 64
- ST\_AsGeoJSON (class in *geoalchemy2.functions*), 64
- ST\_AsGML (class in *geoalchemy2.functions*), 64
- ST\_AsHEXEWKB (class in *geoalchemy2.functions*), 64
- ST\_AsHexWKB (class in *geoalchemy2.functions*), 64
- ST\_AsJPEG (class in *geoalchemy2.functions*), 64
- ST\_AsKML (class in *geoalchemy2.functions*), 64
- ST\_AsLatLonText (class in *geoalchemy2.functions*), 65
- ST\_AsMVT (class in *geoalchemy2.functions*), 65
- ST\_AsMVTGeom (class in *geoalchemy2.functions*), 65
- ST\_Aspect (class in *geoalchemy2.functions*), 66
- ST\_AsPNG (class in *geoalchemy2.functions*), 65
- ST\_AsRaster (class in *geoalchemy2.functions*), 65
- ST\_AsSVG (class in *geoalchemy2.functions*), 65
- ST\_AsText (class in *geoalchemy2.functions*), 65
- ST\_AsTIFF (class in *geoalchemy2.functions*), 65
- ST\_AsTWKB (class in *geoalchemy2.functions*), 65
- ST\_AsWKB (class in *geoalchemy2.functions*), 65
- ST\_AsX3D (class in *geoalchemy2.functions*), 66
- ST\_Azimuth (class in *geoalchemy2.functions*), 66
- ST\_Band (class in *geoalchemy2.functions*), 66
- ST\_BandFileSize (class in *geoalchemy2.functions*), 66
- ST\_BandFileTimestamp (class in *geoalchemy2.functions*), 66
- ST\_BandIsNoData (class in *geoalchemy2.functions*), 66
- ST\_BandMetaData (class in *geoalchemy2.functions*), 66
- ST\_BandNoDataValue (class in *geoalchemy2.functions*), 66
- ST\_BandPath (class in *geoalchemy2.functions*), 66
- ST\_BandPixelType (class in *geoalchemy2.functions*), 67
- ST\_BdMPolyFromText (class in *geoalchemy2.functions*), 67
- ST\_BdPolyFromText (class in *geoalchemy2.functions*), 67
- ST\_Boundary (class in *geoalchemy2.functions*), 67
- ST\_BoundingDiagonal (class in *geoalchemy2.functions*), 67
- ST\_Box2dFromGeoHash (class in *geoalchemy2.functions*), 67
- ST\_Buffer (class in *geoalchemy2.functions*), 67
- ST\_BuildArea (class in *geoalchemy2.functions*), 68
- ST\_Centroid (class in *geoalchemy2.functions*), 68
- ST\_ChaikinSmoothing (class in *geoalchemy2.functions*), 68
- ST\_Clip (class in *geoalchemy2.functions*), 68
- ST\_ClipByBox2D (class in *geoalchemy2.functions*), 68
- ST\_ClosestPoint (class in *geoalchemy2.functions*), 68
- ST\_ClosestPointOfApproach (class in *geoalchemy2.functions*), 69
- ST\_ClusterDBSCAN (class in *geoalchemy2.functions*), 69
- ST\_ClusterIntersecting (class in *geoalchemy2.functions*), 69
- ST\_ClusterKMeans (class in *geoalchemy2.functions*), 69
- ST\_ClusterWithin (class in *geoalchemy2.functions*), 69
- ST\_Collect (class in *geoalchemy2.functions*), 69
- ST\_CollectionExtract (class in *geoalchemy2.functions*), 69
- ST\_CollectionHomogenize (class in *geoalchemy2.functions*), 70
- ST\_ColorMap (class in *geoalchemy2.functions*), 70
- ST\_ConcaveHull (class in *geoalchemy2.functions*), 70
- ST\_ConstrainedDelaunayTriangles (class in *geoalchemy2.functions*), 70
- ST\_Contains (class in *geoalchemy2.functions*), 70
- ST\_ContainsProperly (class in *geoalchemy2.functions*), 70
- ST\_ConvexHull (class in *geoalchemy2.functions*), 70
- ST\_CoordDim (class in *geoalchemy2.functions*), 71
- ST\_Count (class in *geoalchemy2.functions*), 71
- ST\_CountAgg (class in *geoalchemy2.functions*), 71
- ST\_CoveredBy (class in *geoalchemy2.functions*), 71



- ST\_Covers (class in *geoalchemy2.functions*), 71
- ST\_CPAWithin (class in *geoalchemy2.functions*), 68
- ST\_Crosses (class in *geoalchemy2.functions*), 71
- ST\_CurveToLine (class in *geoalchemy2.functions*), 71
- ST\_DelaunayTriangles (class in *geoalchemy2.functions*), 72
- ST\_DFullyWithin (class in *geoalchemy2.functions*), 71
- ST\_Difference (class in *geoalchemy2.functions*), 72
- ST\_Dimension (class in *geoalchemy2.functions*), 72
- ST\_Disjoint (class in *geoalchemy2.functions*), 72
- ST\_Distance (class in *geoalchemy2.functions*), 72
- ST\_Distance\_Sphere (class in *geoalchemy2.functions*), 72
- ST\_DistanceCPA (class in *geoalchemy2.functions*), 72
- ST\_DistanceSphere (class in *geoalchemy2.functions*), 72
- ST\_DistanceSpheroid (class in *geoalchemy2.functions*), 72
- ST\_Distinct4ma (class in *geoalchemy2.functions*), 73
- ST\_Dump (class in *geoalchemy2.functions*), 73
- ST\_DumpAsPolygons (class in *geoalchemy2.functions*), 73
- ST\_DumpPoints (class in *geoalchemy2.functions*), 73
- ST\_DumpRings (class in *geoalchemy2.functions*), 73
- ST\_DumpValues (class in *geoalchemy2.functions*), 73
- ST\_DWithin (class in *geoalchemy2.functions*), 71
- ST\_EndPoint (class in *geoalchemy2.functions*), 73
- ST\_Envelope (class in *geoalchemy2.functions*), 73
- ST\_Equals (class in *geoalchemy2.functions*), 74
- ST\_EstimatedExtent (class in *geoalchemy2.functions*), 74
- ST\_Expand (class in *geoalchemy2.functions*), 74
- ST\_Extent (class in *geoalchemy2.functions*), 74
- ST\_ExteriorRing (class in *geoalchemy2.functions*), 74
- ST\_Extrude (class in *geoalchemy2.functions*), 74
- ST\_FilterByM (class in *geoalchemy2.functions*), 74
- ST\_FlipCoordinates (class in *geoalchemy2.functions*), 75
- ST\_Force2D (class in *geoalchemy2.functions*), 75
- ST\_Force3D (class in *geoalchemy2.functions*), 75
- ST\_Force3DM (class in *geoalchemy2.functions*), 75
- ST\_Force3DZ (class in *geoalchemy2.functions*), 75
- ST\_Force4D (class in *geoalchemy2.functions*), 75
- ST\_ForceCollection (class in *geoalchemy2.functions*), 76
- ST\_ForceCurve (class in *geoalchemy2.functions*), 76
- ST\_ForceLHR (class in *geoalchemy2.functions*), 76
- ST\_ForcePolygonCCW (class in *geoalchemy2.functions*), 76
- ST\_ForcePolygonCW (class in *geoalchemy2.functions*), 76
- ST\_ForceRHR (class in *geoalchemy2.functions*), 76
- ST\_ForceSFS (class in *geoalchemy2.functions*), 76
- ST\_FrechetDistance (class in *geoalchemy2.functions*), 77
- ST\_FromGDALRaster (class in *geoalchemy2.functions*), 77
- ST\_GeneratePoints (class in *geoalchemy2.functions*), 77
- ST\_GeogFromText (class in *geoalchemy2.functions*), 77
- ST\_GeogFromWKB (class in *geoalchemy2.functions*), 78
- ST\_GeographyFromText (class in *geoalchemy2.functions*), 78
- ST\_GeoHash (class in *geoalchemy2.functions*), 77
- ST\_GeomCollFromText (class in *geoalchemy2.functions*), 78
- ST\_GeometricMedian (class in *geoalchemy2.functions*), 79
- ST\_GeometryFromText (class in *geoalchemy2.functions*), 80
- ST\_GeometryN (class in *geoalchemy2.functions*), 80
- ST\_GeometryType (class in *geoalchemy2.functions*), 80
- ST\_GeomFromEWKB (class in *geoalchemy2.functions*), 78
- ST\_GeomFromEWKT (class in *geoalchemy2.functions*), 78
- ST\_GeomFromGeoHash (class in *geoalchemy2.functions*), 79
- ST\_GeomFromGeoJSON (class in *geoalchemy2.functions*), 79
- ST\_GeomFromGML (class in *geoalchemy2.functions*), 78
- ST\_GeomFromKML (class in *geoalchemy2.functions*), 79
- ST\_GeomFromText (class in *geoalchemy2.functions*), 79
- ST\_GeomFromTWKB (class in *geoalchemy2.functions*), 79
- ST\_GeomFromWKB (class in *geoalchemy2.functions*), 79
- ST\_GeoReference (class in *geoalchemy2.functions*), 77
- ST\_GMLToSQL (class in *geoalchemy2.functions*), 77
- ST\_Grayscale (class in *geoalchemy2.functions*), 80
- ST\_HasArc (class in *geoalchemy2.functions*), 80
- ST\_HasNoBand (class in *geoalchemy2.functions*), 80
- ST\_HausdorffDistance (class in *geoalchemy2.functions*), 80
- ST\_Height (class in *geoalchemy2.functions*), 80
- ST\_HillShade (class in *geoalchemy2.functions*), 81
- ST\_Histogram (class in *geoalchemy2.functions*), 81
- ST\_InteriorRingN (class in *geoalchemy2.functions*), 81
- ST\_InterpolatePoint (class in *geoalchemy2.functions*), 81
- ST\_Intersection (class in *geoalchemy2.functions*), 81
- ST\_Intersects (class in *geoalchemy2.functions*), 81
- ST\_InvDistWeight4ma (class in *geoalchemy2.functions*), 81
- ST\_IsClosed (class in *geoalchemy2.functions*), 81
- ST\_IsCollection (class in *geoalchemy2.functions*), 82
- ST\_IsEmpty (class in *geoalchemy2.functions*), 82
- ST\_IsPlanar (class in *geoalchemy2.functions*), 82
- ST\_IsPolygonCCW (class in *geoalchemy2.functions*), 82
- ST\_IsPolygonCW (class in *geoalchemy2.functions*), 82
- ST\_IsRing (class in *geoalchemy2.functions*), 82

`ST_IsSimple` (class in `geoalchemy2.functions`), 82  
`ST_IsSolid` (class in `geoalchemy2.functions`), 82  
`ST_IsValid` (class in `geoalchemy2.functions`), 82  
`ST_IsValidDetail` (class in `geoalchemy2.functions`), 82  
`ST_IsValidReason` (class in `geoalchemy2.functions`), 82  
`ST_IsValidTrajectory` (class in `geoalchemy2.functions`), 82  
`ST_Length` (class in `geoalchemy2.functions`), 83  
`ST_Length2D` (class in `geoalchemy2.functions`), 83  
`ST_LengthSpheroid` (class in `geoalchemy2.functions`), 83  
`ST_LineCrossingDirection` (class in `geoalchemy2.functions`), 83  
`ST_LineFromEncodedPolyline` (class in `geoalchemy2.functions`), 83  
`ST_LineFromMultiPoint` (class in `geoalchemy2.functions`), 83  
`ST_LineFromText` (class in `geoalchemy2.functions`), 83  
`ST_LineFromWKB` (class in `geoalchemy2.functions`), 83  
`ST_LineInterpolatePoint` (class in `geoalchemy2.functions`), 84  
`ST_LineInterpolatePoints` (class in `geoalchemy2.functions`), 84  
`ST_LineLocatePoint` (class in `geoalchemy2.functions`), 84  
`ST_LineMerge` (class in `geoalchemy2.functions`), 84  
`ST_LinestringFromWKB` (class in `geoalchemy2.functions`), 84  
`ST_LineSubstring` (class in `geoalchemy2.functions`), 84  
`ST_LineToCurve` (class in `geoalchemy2.functions`), 84  
`ST_LocateAlong` (class in `geoalchemy2.functions`), 85  
`ST_LocateBetween` (class in `geoalchemy2.functions`), 85  
`ST_LocateBetweenElevations` (class in `geoalchemy2.functions`), 85  
`ST_LongestLine` (class in `geoalchemy2.functions`), 85  
`ST_M` (class in `geoalchemy2.functions`), 85  
`ST_MakeBox2D` (class in `geoalchemy2.functions`), 86  
`ST_MakeEmptyCoverage` (class in `geoalchemy2.functions`), 86  
`ST_MakeEmptyRaster` (class in `geoalchemy2.functions`), 86  
`ST_MakeEnvelope` (class in `geoalchemy2.functions`), 86  
`ST_MakeLine` (class in `geoalchemy2.functions`), 86  
`ST_MakePoint` (class in `geoalchemy2.functions`), 87  
`ST_MakePointM` (class in `geoalchemy2.functions`), 87  
`ST_MakePolygon` (class in `geoalchemy2.functions`), 87  
`ST_MakeSolid` (class in `geoalchemy2.functions`), 87  
`ST_MakeValid` (class in `geoalchemy2.functions`), 87  
`ST_MapAlgebra` (class in `geoalchemy2.functions`), 87  
`ST_MapAlgebraExpr` (class in `geoalchemy2.functions`), 88  
`ST_MapAlgebraFct` (class in `geoalchemy2.functions`), 88  
`ST_MapAlgebraFctNgb` (class in `geoalchemy2.functions`), 88  
`ST_Max4ma` (class in `geoalchemy2.functions`), 88  
`ST_MaxDistance` (class in `geoalchemy2.functions`), 88  
`ST_Mean4ma` (class in `geoalchemy2.functions`), 88  
`ST_MemSize` (class in `geoalchemy2.functions`), 89  
`ST_MemUnion` (class in `geoalchemy2.functions`), 89  
`ST_MetaData` (class in `geoalchemy2.functions`), 89  
`ST_Min4ma` (class in `geoalchemy2.functions`), 89  
`ST_MinConvexHull` (class in `geoalchemy2.functions`), 89  
`ST_MinDist4ma` (class in `geoalchemy2.functions`), 89  
`ST_MinimumBoundingCircle` (class in `geoalchemy2.functions`), 89  
`ST_MinimumBoundingRadius` (class in `geoalchemy2.functions`), 89  
`ST_MinimumClearance` (class in `geoalchemy2.functions`), 90  
`ST_MinimumClearanceLine` (class in `geoalchemy2.functions`), 90  
`ST_MinkowskiSum` (class in `geoalchemy2.functions`), 90  
`ST_MinPossibleValue` (class in `geoalchemy2.functions`), 89  
`ST_MLineFromText` (class in `geoalchemy2.functions`), 85  
`ST_MPointFromText` (class in `geoalchemy2.functions`), 86  
`ST_MPolyFromText` (class in `geoalchemy2.functions`), 86  
`ST_Multi` (class in `geoalchemy2.functions`), 90  
`ST_NDims` (class in `geoalchemy2.functions`), 90  
`ST_NearestValue` (class in `geoalchemy2.functions`), 90  
`ST_Neighborhood` (class in `geoalchemy2.functions`), 90  
`ST_Node` (class in `geoalchemy2.functions`), 91  
`ST_Normalize` (class in `geoalchemy2.functions`), 91  
`ST_NotSameAlignmentReason` (class in `geoalchemy2.functions`), 91  
`ST_NPoints` (class in `geoalchemy2.functions`), 90  
`ST_NRings` (class in `geoalchemy2.functions`), 90  
`ST_NumBands` (class in `geoalchemy2.functions`), 91  
`ST_NumGeometries` (class in `geoalchemy2.functions`), 91  
`ST_NumInteriorRing` (class in `geoalchemy2.functions`), 91  
`ST_NumInteriorRings` (class in `geoalchemy2.functions`), 91  
`ST_NumPatches` (class in `geoalchemy2.functions`), 91  
`ST_NumPoints` (class in `geoalchemy2.functions`), 91  
`ST_OffsetCurve` (class in `geoalchemy2.functions`), 91  
`ST_OrderingEquals` (class in `geoalchemy2.functions`), 92  
`ST_Orientation` (class in `geoalchemy2.functions`), 92  
`ST_OrientedEnvelope` (class in `geoalchemy2.functions`), 92  
`ST_Overlaps` (class in `geoalchemy2.functions`), 92  
`ST_PatchN` (class in `geoalchemy2.functions`), 92  
`ST_Perimeter` (class in `geoalchemy2.functions`), 92  
`ST_Perimeter2D` (class in `geoalchemy2.functions`), 92  
`ST_PixelAsCentroid` (class in `geoalchemy2.functions`), 92

- ST\_PixelAsCentroids (class in *geoalchemy2.functions*), 93
- ST\_PixelAsPoint (class in *geoalchemy2.functions*), 93
- ST\_PixelAsPoints (class in *geoalchemy2.functions*), 93
- ST\_PixelAsPolygon (class in *geoalchemy2.functions*), 93
- ST\_PixelAsPolygons (class in *geoalchemy2.functions*), 93
- ST\_PixelHeight (class in *geoalchemy2.functions*), 93
- ST\_PixelOfValue (class in *geoalchemy2.functions*), 93
- ST\_PixelWidth (class in *geoalchemy2.functions*), 93
- ST\_Point (class in *geoalchemy2.functions*), 93
- ST\_PointFromGeoHash (class in *geoalchemy2.functions*), 94
- ST\_PointFromText (class in *geoalchemy2.functions*), 94
- ST\_PointFromWKB (class in *geoalchemy2.functions*), 94
- ST\_PointInsideCircle (class in *geoalchemy2.functions*), 94
- ST\_PointN (class in *geoalchemy2.functions*), 94
- ST\_PointOnSurface (class in *geoalchemy2.functions*), 94
- ST\_Points (class in *geoalchemy2.functions*), 94
- ST\_Polygon (class in *geoalchemy2.functions*), 95
- ST\_PolygonFromText (class in *geoalchemy2.functions*), 95
- ST\_Polygonize (class in *geoalchemy2.functions*), 95
- ST\_Project (class in *geoalchemy2.functions*), 95
- ST\_Quantile (class in *geoalchemy2.functions*), 95
- ST\_QuantizeCoordinates (class in *geoalchemy2.functions*), 95
- ST\_Range4ma (class in *geoalchemy2.functions*), 95
- ST\_RasterToWorldCoord (class in *geoalchemy2.functions*), 96
- ST\_RasterToWorldCoordX (class in *geoalchemy2.functions*), 96
- ST\_RasterToWorldCoordY (class in *geoalchemy2.functions*), 96
- ST\_RastFromHexWKB (class in *geoalchemy2.functions*), 96
- ST\_RastFromWKB (class in *geoalchemy2.functions*), 96
- ST\_Reclass (class in *geoalchemy2.functions*), 96
- ST\_Relate (class in *geoalchemy2.functions*), 96
- ST\_RelateMatch (class in *geoalchemy2.functions*), 96
- ST\_RemovePoint (class in *geoalchemy2.functions*), 97
- ST\_RemoveRepeatedPoints (class in *geoalchemy2.functions*), 97
- ST\_Resample (class in *geoalchemy2.functions*), 97
- ST\_Rescale (class in *geoalchemy2.functions*), 97
- ST\_Resize (class in *geoalchemy2.functions*), 97
- ST\_Reskew (class in *geoalchemy2.functions*), 97
- ST\_Retile (class in *geoalchemy2.functions*), 98
- ST\_Reverse (class in *geoalchemy2.functions*), 98
- ST\_Rotate (class in *geoalchemy2.functions*), 98
- ST\_RotateX (class in *geoalchemy2.functions*), 98
- ST\_RotateY (class in *geoalchemy2.functions*), 98
- ST\_RotateZ (class in *geoalchemy2.functions*), 98
- ST\_Rotation (class in *geoalchemy2.functions*), 99
- ST\_Roughness (class in *geoalchemy2.functions*), 99
- ST\_SameAlignment (class in *geoalchemy2.functions*), 99
- ST\_Scale (class in *geoalchemy2.functions*), 99
- ST\_ScaleX (class in *geoalchemy2.functions*), 99
- ST\_ScaleY (class in *geoalchemy2.functions*), 99
- ST\_Segmentize (class in *geoalchemy2.functions*), 99
- ST\_SetBandIndex (class in *geoalchemy2.functions*), 99
- ST\_SetBandIsNoData (class in *geoalchemy2.functions*), 100
- ST\_SetBandNoDataValue (class in *geoalchemy2.functions*), 100
- ST\_SetBandPath (class in *geoalchemy2.functions*), 100
- ST\_SetEffectiveArea (class in *geoalchemy2.functions*), 100
- ST\_SetGeoReference (class in *geoalchemy2.functions*), 100
- ST\_SetPoint (class in *geoalchemy2.functions*), 100
- ST\_SetRotation (class in *geoalchemy2.functions*), 101
- ST\_SetScale (class in *geoalchemy2.functions*), 101
- ST\_SetSkew (class in *geoalchemy2.functions*), 101
- ST\_SetSRID (class in *geoalchemy2.functions*), 101
- ST\_SetUpperLeft (class in *geoalchemy2.functions*), 101
- ST\_SetValue (class in *geoalchemy2.functions*), 101
- ST\_SetValues (class in *geoalchemy2.functions*), 102
- ST\_SharedPaths (class in *geoalchemy2.functions*), 102
- ST\_ShiftLongitude (class in *geoalchemy2.functions*), 102
- ST\_ShortestLine (class in *geoalchemy2.functions*), 102
- ST\_Simplify (class in *geoalchemy2.functions*), 102
- ST\_SimplifyPreserveTopology (class in *geoalchemy2.functions*), 102
- ST\_SimplifyVW (class in *geoalchemy2.functions*), 103
- ST\_SkewX (class in *geoalchemy2.functions*), 103
- ST\_SkewY (class in *geoalchemy2.functions*), 103
- ST\_Slope (class in *geoalchemy2.functions*), 103
- ST\_Snap (class in *geoalchemy2.functions*), 103
- ST\_SnapToGrid (class in *geoalchemy2.functions*), 103
- ST\_Split (class in *geoalchemy2.functions*), 103
- ST\_SRID (class in *geoalchemy2.functions*), 99
- ST\_StartPoint (class in *geoalchemy2.functions*), 104
- ST\_StdDev4ma (class in *geoalchemy2.functions*), 104
- ST\_StraightSkeleton (class in *geoalchemy2.functions*), 104
- ST\_Subdivide (class in *geoalchemy2.functions*), 104
- ST\_Sum4ma (class in *geoalchemy2.functions*), 104
- ST\_Summary (class in *geoalchemy2.functions*), 104
- ST\_SummaryStats (class in *geoalchemy2.functions*), 104
- ST\_SummaryStatsAgg (class in *geoalchemy2.functions*), 104
- ST\_SwapOrdinates (class in *geoalchemy2.functions*), 105



ST_SymDifference (class in <i>geoalchemy2.functions</i> ), 105	type ( <i>geoalchemy2.functions.PostGIS_DropBBox</i> attribute), 59
ST_Tessellate (class in <i>geoalchemy2.functions</i> ), 105	type ( <i>geoalchemy2.functions.ST_3DClosestPoint</i> attribute), 61
ST_Tile (class in <i>geoalchemy2.functions</i> ), 105	type ( <i>geoalchemy2.functions.ST_3DDifference</i> attribute), 61
ST_TileEnvelope (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_3DIntersection</i> attribute), 61
ST_Touches (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_3DLineInterpolatePoint</i> attribute), 62
ST_TPI (class in <i>geoalchemy2.functions</i> ), 105	type ( <i>geoalchemy2.functions.ST_3DLongestLine</i> attribute), 62
ST_Transform (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_3DShortestLine</i> attribute), 62
ST_Translate (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_3DUnion</i> attribute), 62
ST_TransScale (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_AddBand</i> attribute), 63
ST_TRI (class in <i>geoalchemy2.functions</i> ), 105	type ( <i>geoalchemy2.functions.ST_AddMeasure</i> attribute), 63
ST_UnaryUnion (class in <i>geoalchemy2.functions</i> ), 106	type ( <i>geoalchemy2.functions.ST_AddPoint</i> attribute), 63
ST_Union (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_Affine</i> attribute), 63
ST_UpperLeftX (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_ApproximateMedialAxis</i> attribute), 63
ST_UpperLeftY (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_AsMVTGeom</i> attribute), 65
ST_Value (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_Aspect</i> attribute), 66
ST_ValueCount (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_AsRaster</i> attribute), 65
ST_Volume (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_Band</i> attribute), 66
ST_VoronoiLines (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_BdMPolyFromText</i> attribute), 67
ST_VoronoiPolygons (class in <i>geoalchemy2.functions</i> ), 107	type ( <i>geoalchemy2.functions.ST_BdPolyFromText</i> attribute), 67
ST_Width (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_Boundary</i> attribute), 67
ST_Within (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_BoundingDiagonal</i> attribute), 67
ST_WKBToSQL (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_Buffer</i> attribute), 67
ST_WKTToSQL (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_BuildArea</i> attribute), 68
ST_WorldToRasterCoord (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_Centroid</i> attribute), 68
ST_WorldToRasterCoordX (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_ChaikinSmoothing</i> attribute), 68
ST_WorldToRasterCoordY (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_Clip</i> attribute), 68
ST_WrapX (class in <i>geoalchemy2.functions</i> ), 108	type ( <i>geoalchemy2.functions.ST_ClipByBox2D</i> attribute), 68
ST_X (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_ClosestPoint</i> attribute), 69
ST_XMax (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_ClusterIntersecting</i> attribute), 69
ST_XMin (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_ClusterWithin</i> attribute), 69
ST_Y (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_Collect</i> attribute), 69
ST_YMax (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_CollectionExtract</i> attribute), 69
ST_YMin (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_CollectionHomogenize</i> attribute), 70
ST_Z (class in <i>geoalchemy2.functions</i> ), 109	type ( <i>geoalchemy2.functions.ST_ColorMap</i> attribute),
ST_ZMax (class in <i>geoalchemy2.functions</i> ), 109	
ST_Zmflag (class in <i>geoalchemy2.functions</i> ), 109	
ST_ZMin (class in <i>geoalchemy2.functions</i> ), 109	
SummaryStats (class in <i>geoalchemy2.types</i> ), 53	
<b>T</b>	
TableRowElement (class in <i>geoalchemy2.functions</i> ), 109	
to_left() ( <i>geoalchemy2.comparator.Comparator</i> method), 112	
to_right() ( <i>geoalchemy2.comparator.Comparator</i> method), 112	
to_shape() (in module <i>geoalchemy2.shape</i> ), 113	
type ( <i>geoalchemy2.functions.PostGIS_AddBBox</i> attribute), 59	



70		tribute), 78	
type (geoalchemy2.functions.ST_ConcaveHull attribute), 70	type (geoalchemy2.functions.ST_GeographyFromText attribute), 78		
type (geoalchemy2.functions.ST_ConstrainedDelaunayTriangulation attribute), 70	type (geoalchemy2.functions.ST_GeomCollFromText attribute), 78		
type (geoalchemy2.functions.ST_ConvexHull attribute), 71	type (geoalchemy2.functions.ST_GeometricMedian attribute), 80		
type (geoalchemy2.functions.ST_CurveToLine attribute), 71	type (geoalchemy2.functions.ST_GeometryFromText attribute), 80		
type (geoalchemy2.functions.ST_DelaunayTriangles attribute), 72	type (geoalchemy2.functions.ST_GeometryN attribute), 80		
type (geoalchemy2.functions.ST_Difference attribute), 72	type (geoalchemy2.functions.ST_GeomFromEWKB attribute), 78		
type (geoalchemy2.functions.ST_Dump attribute), 73	type (geoalchemy2.functions.ST_GeomFromEWKT attribute), 78		
type (geoalchemy2.functions.ST_DumpPoints attribute), 73	type (geoalchemy2.functions.ST_GeomFromGeoHash attribute), 79		
type (geoalchemy2.functions.ST_DumpRings attribute), 73	type (geoalchemy2.functions.ST_GeomFromGeoJSON attribute), 79		
type (geoalchemy2.functions.ST_EndPoint attribute), 73	type (geoalchemy2.functions.ST_GeomFromGML attribute), 78		
type (geoalchemy2.functions.ST_Envelope attribute), 74	type (geoalchemy2.functions.ST_GeomFromKML attribute), 79		
type (geoalchemy2.functions.ST_Expand attribute), 74	type (geoalchemy2.functions.ST_GeomFromText attribute), 79		
type (geoalchemy2.functions.ST_ExteriorRing attribute), 74	type (geoalchemy2.functions.ST_GeomFromTWKB attribute), 79		
type (geoalchemy2.functions.ST_Extrude attribute), 74	type (geoalchemy2.functions.ST_GeomFromWKB attribute), 79		
type (geoalchemy2.functions.ST_FilterByM attribute), 74	type (geoalchemy2.functions.ST_GMLToSQL attribute), 77		
type (geoalchemy2.functions.ST_FlipCoordinates attribute), 75	type (geoalchemy2.functions.ST_Grayscale attribute), 80		
type (geoalchemy2.functions.ST_Force2D attribute), 75	type (geoalchemy2.functions.ST_HillShade attribute), 81		
type (geoalchemy2.functions.ST_Force3D attribute), 75	type (geoalchemy2.functions.ST_InteriorRingN attribute), 81		
type (geoalchemy2.functions.ST_Force3DM attribute), 75	type (geoalchemy2.functions.ST_Intersection attribute), 81		
type (geoalchemy2.functions.ST_Force3DZ attribute), 75	type (geoalchemy2.functions.ST_LineFromEncodedPolyline attribute), 83		
type (geoalchemy2.functions.ST_Force4D attribute), 75	type (geoalchemy2.functions.ST_LineFromMultiPoint attribute), 83		
type (geoalchemy2.functions.ST_ForceCollection attribute), 76	type (geoalchemy2.functions.ST_LineFromText attribute), 83		
type (geoalchemy2.functions.ST_ForceCurve attribute), 76	type (geoalchemy2.functions.ST_LineFromWKB attribute), 83		
type (geoalchemy2.functions.ST_ForceLHR attribute), 76	type (geoalchemy2.functions.ST_LineInterpolatePoint attribute), 84		
type (geoalchemy2.functions.ST_ForcePolygonCCW attribute), 76	type (geoalchemy2.functions.ST_LineInterpolatePoints attribute), 84		
type (geoalchemy2.functions.ST_ForcePolygonCW attribute), 76	type (geoalchemy2.functions.ST_LineMerge attribute), 84		
type (geoalchemy2.functions.ST_ForceRHR attribute), 76	type (geoalchemy2.functions.ST_LinestringFromWKB attribute), 85		
type (geoalchemy2.functions.ST_ForceSFS attribute), 77	type (geoalchemy2.functions.ST_LineSubstring attribute), 85		
type (geoalchemy2.functions.ST_FromGDALRaster attribute), 77			
type (geoalchemy2.functions.ST_GeneratePoints attribute), 77			
type (geoalchemy2.functions.ST_GeogFromText attribute), 77			
type (geoalchemy2.functions.ST_GeogFromWKB attribute), 77			

tribute), 84  
type (geoalchemy2.functions.ST\_LineToCurve attribute), 84  
type (geoalchemy2.functions.ST\_LocateAlong attribute), 85  
type (geoalchemy2.functions.ST\_LocateBetween attribute), 85  
type (geoalchemy2.functions.ST\_LocateBetweenElevations attribute), 85  
type (geoalchemy2.functions.ST\_LongestLine attribute), 85  
type (geoalchemy2.functions.ST\_MakeEmptyCoverage attribute), 86  
type (geoalchemy2.functions.ST\_MakeEmptyRaster attribute), 86  
type (geoalchemy2.functions.ST\_MakeEnvelope attribute), 86  
type (geoalchemy2.functions.ST\_MakeLine attribute), 87  
type (geoalchemy2.functions.ST\_MakePoint attribute), 87  
type (geoalchemy2.functions.ST\_MakePointM attribute), 87  
type (geoalchemy2.functions.ST\_MakePolygon attribute), 87  
type (geoalchemy2.functions.ST\_MakeSolid attribute), 87  
type (geoalchemy2.functions.ST\_MakeValid attribute), 87  
type (geoalchemy2.functions.ST\_MapAlgebraExpr attribute), 88  
type (geoalchemy2.functions.ST\_MapAlgebraFct attribute), 88  
type (geoalchemy2.functions.ST\_MapAlgebraFctNgb attribute), 88  
type (geoalchemy2.functions.ST\_MemUnion attribute), 89  
type (geoalchemy2.functions.ST\_MinConvexHull attribute), 89  
type (geoalchemy2.functions.ST\_MinimumBoundingCircle attribute), 89  
type (geoalchemy2.functions.ST\_MinimumClearanceLine attribute), 90  
type (geoalchemy2.functions.ST\_MinkowskiSum attribute), 90  
type (geoalchemy2.functions.ST\_MLineFromText attribute), 85  
type (geoalchemy2.functions.ST\_MPointFromText attribute), 86  
type (geoalchemy2.functions.ST\_MPolyFromText attribute), 86  
type (geoalchemy2.functions.ST\_Multi attribute), 90  
type (geoalchemy2.functions.ST\_Node attribute), 91  
type (geoalchemy2.functions.ST\_Normalize attribute), 91  
type (geoalchemy2.functions.ST\_OffsetCurve attribute), 92  
type (geoalchemy2.functions.ST\_OrientedEnvelope attribute), 92  
type (geoalchemy2.functions.ST\_PatchN attribute), 92  
type (geoalchemy2.functions.ST\_PixelAsCentroid attribute), 92  
type (geoalchemy2.functions.ST\_PixelAsPoint attribute), 93  
type (geoalchemy2.functions.ST\_PixelAsPolygon attribute), 93  
type (geoalchemy2.functions.ST\_Point attribute), 93  
type (geoalchemy2.functions.ST\_PointFromGeoHash attribute), 94  
type (geoalchemy2.functions.ST\_PointFromText attribute), 94  
type (geoalchemy2.functions.ST\_PointFromWKB attribute), 94  
type (geoalchemy2.functions.ST\_PointN attribute), 94  
type (geoalchemy2.functions.ST\_PointOnSurface attribute), 94  
type (geoalchemy2.functions.ST\_Points attribute), 94  
type (geoalchemy2.functions.ST\_Polygon attribute), 95  
type (geoalchemy2.functions.ST\_PolygonFromText attribute), 95  
type (geoalchemy2.functions.ST\_Polygonize attribute), 95  
type (geoalchemy2.functions.ST\_Project attribute), 95  
type (geoalchemy2.functions.ST\_QuantizeCoordinates attribute), 95  
type (geoalchemy2.functions.ST\_RastFromHexWKB attribute), 96  
type (geoalchemy2.functions.ST\_RastFromWKB attribute), 96  
type (geoalchemy2.functions.ST\_Reclass attribute), 96  
type (geoalchemy2.functions.ST\_RemovePoint attribute), 97  
type (geoalchemy2.functions.ST\_RemoveRepeatedPoints attribute), 97  
type (geoalchemy2.functions.ST\_Resample attribute), 97  
type (geoalchemy2.functions.ST\_Rescale attribute), 97  
type (geoalchemy2.functions.ST\_Resize attribute), 97  
type (geoalchemy2.functions.ST\_Reskew attribute), 98  
type (geoalchemy2.functions.ST\_Retile attribute), 98  
type (geoalchemy2.functions.ST\_Reverse attribute), 98  
type (geoalchemy2.functions.ST\_Rotate attribute), 98  
type (geoalchemy2.functions.ST\_RotateX attribute), 98  
type (geoalchemy2.functions.ST\_RotateY attribute), 98  
type (geoalchemy2.functions.ST\_RotateZ attribute), 98  
type (geoalchemy2.functions.ST\_Roughness attribute), 99  
type (geoalchemy2.functions.ST\_Scale attribute), 99  
type (geoalchemy2.functions.ST\_Segmentize attribute), 99

- type (*geoalchemy2.functions.ST\_SetBandIndex attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetBandIsNoData attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetBandNoDataValue attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetBandPath attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetEffectiveArea attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetGeoReference attribute*), 100
  - type (*geoalchemy2.functions.ST\_SetPoint attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetRotation attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetScale attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetSkew attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetSRID attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetUpperLeft attribute*), 101
  - type (*geoalchemy2.functions.ST\_SetValue attribute*), 102
  - type (*geoalchemy2.functions.ST\_SetValues attribute*), 102
  - type (*geoalchemy2.functions.ST\_SharedPaths attribute*), 102
  - type (*geoalchemy2.functions.ST\_ShiftLongitude attribute*), 102
  - type (*geoalchemy2.functions.ST\_ShortestLine attribute*), 102
  - type (*geoalchemy2.functions.ST\_Simplify attribute*), 102
  - type (*geoalchemy2.functions.ST\_SimplifyPreserveTopology attribute*), 103
  - type (*geoalchemy2.functions.ST\_SimplifyVW attribute*), 103
  - type (*geoalchemy2.functions.ST\_Slope attribute*), 103
  - type (*geoalchemy2.functions.ST\_Snap attribute*), 103
  - type (*geoalchemy2.functions.ST\_SnapToGrid attribute*), 103
  - type (*geoalchemy2.functions.ST\_Split attribute*), 104
  - type (*geoalchemy2.functions.ST\_StartPoint attribute*), 104
  - type (*geoalchemy2.functions.ST\_StraightSkeleton attribute*), 104
  - type (*geoalchemy2.functions.ST\_Subdivide attribute*), 104
  - type (*geoalchemy2.functions.ST\_SummaryStatsAgg attribute*), 105
  - type (*geoalchemy2.functions.ST\_SwapOrdinates attribute*), 105
  - type (*geoalchemy2.functions.ST\_SymDifference attribute*), 105
  - type (*geoalchemy2.functions.ST\_Tessellate attribute*), 105
  - type (*geoalchemy2.functions.ST\_Tile attribute*), 106
  - type (*geoalchemy2.functions.ST\_TileEnvelope attribute*), 106
  - type (*geoalchemy2.functions.ST\_TPI attribute*), 105
  - type (*geoalchemy2.functions.ST\_Transform attribute*), 106
  - type (*geoalchemy2.functions.ST\_Translate attribute*), 106
  - type (*geoalchemy2.functions.ST\_TransScale attribute*), 106
  - type (*geoalchemy2.functions.ST\_TRI attribute*), 105
  - type (*geoalchemy2.functions.ST\_UnaryUnion attribute*), 107
  - type (*geoalchemy2.functions.ST\_Union attribute*), 107
  - type (*geoalchemy2.functions.ST\_VoronoiLines attribute*), 107
  - type (*geoalchemy2.functions.ST\_VoronoiPolygons attribute*), 108
  - type (*geoalchemy2.functions.ST\_WKBToSQL attribute*), 108
  - type (*geoalchemy2.functions.ST\_WKTToSQL attribute*), 108
  - type (*geoalchemy2.functions.ST\_WrapX attribute*), 109
  - typemap (*geoalchemy2.types.CompositeType attribute*), 51
  - typemap (*geoalchemy2.types.GeometryDump attribute*), 52
  - typemap (*geoalchemy2.types.SummaryStats attribute*), 53
- ## U
- UnlockRows (*class in geoalchemy2.functions*), 109
  - UpdateGeometrySRID (*class in geoalchemy2.functions*), 110
- ## V
- visit\_drop\_geospatial\_table() (*in module geoalchemy2.alembic\_helpers*), 117
  - visit\_rename\_geospatial\_table() (*in module geoalchemy2.alembic\_helpers*), 117
- ## W
- WKBElement (*class in geoalchemy2.elements*), 56
  - WKTElement (*class in geoalchemy2.elements*), 56
  - writer (*in module geoalchemy2.alembic\_helpers*), 117