Аксиома.ГИС API для Python

Версия 3.0.0-beta

ООО ЭСТИ

13 апреля, 2021

Оглавление

1 О компоненте

1.1 API для Аксиомы.ГИС на языке Python

API - программный интерфейс приложения - совокупность классов, процедур и функций, благодаря которым одна компьютерная программа может взаимодействовать с другой программой.

С помощью API для Аксиомы.ГИС на языке Python можно взаимодействовать с Аксиомой.ГИС, используя ее функционал для решения задач. Далее понятия «API для Аксиомы.ГИС на языке Python», «Аксиома.ГИС для Python», «Аксиома API» и просто «API» будут относиться к действительно описываемой программной библиотеке.

API использует PySide2, он же Qt for Python. Он идеально подходит для коммерческого использования. Любая ваша разработка с его использованием будет совместима с LGPL и её можно лицензировать на свое усмотрение.

Примечание: Документация API состоит из двух частей:

- *руководство разработчика*;
- *справочник функций*.

1.2 Как читать эту документацию

Данная документация составлена таким образом, чтобы описать общие принципы и подходы решения тех или иных базовых задач, необходимых при обработке геопространственных данных. Более полное описание всевозможных классов, свойств, методов, исключений, функций и их параметров содержится в *справочнике функций*.

Документ логически структурирован. Рекомендуется читать его в прямом порядке от начала до конца, чтобы получить общее представление о возможностях, предоставляемых API.

2 Инициализация

API может быть использован следующими способами:

- в приложении Аксиома.ГИС из панели «Консоль Python»;
- в модуле, который будет загружен в Аксиоме.ГИС;
- как SDK - Software development kit, независимо от приложения Аксиома.ГИС.

Примечание: Для использования API в качестве SDK требуется платная лицензия.

В первых двух случаях можно сразу приступать к работе, так как API будет инициализировано за вас. В последнем случае перед началом использования его необходимо самостоятельно инициализировать. Если забыть это сделать, то при попытке использования будет вызвано исключение, которое напомнит выполнить инициализацию.

Например:

```
from axipy.cs import CoordSystem


try:
    crs = CoordSystem.from_epsg(4326)
except RuntimeError as error:
    print(f"Поймано исключение: {error}")
```

```
>>> Поймано исключение: axipy is not initialized
```

Для инициализации API следует вызвать метод *axipy.init_axioma()*:

```
from axipy import init_axioma
init_axioma()  # инициализация

from axipy.cs import CoordSystem
crs = CoordSystem.from_epsg(4326)
crs.name
```

```
>>> 'Долгота / Широта (WGS 84)'
```

Можно импортировать сразу все классы и функции, чтобы упростить процедуру импорта.

```
from axipy import *  # импортируем все типы Аксиомы
                     # в текущее пространство имен

crs = CoordSystem.from_epsg(4088)
crs.name
```

```
>>> 'World Equidistant Cylindrical (Sphere)'
```

3

Системы Координат

Термины «проекция» и «координатная система» иногда используются один вместо другого, но, на самом деле, понятия, которые они отражают, различны.

Проекция – это уравнения или наборы уравнений, которые содержат математические параметры для карты. Точное число и природа параметров зависят от типа проекции. Проекция – это метод уменьшения искажений карты, вызванных кривизной земной поверхности или, точнее говоря, проекция компенсирует недостатки отображения карты на плоскости в двух измерениях, в то время как координаты существуют в трёх измерениях.

Координатная система – когда параметрам проекции присваиваются определенные значения, они становятся системой координат. Система координат – это набор параметров, описывающих координаты, одна из которых является проекцией.

Системы Координат (СК) представлены типом *axipy.cs.CoordSystem*. Объекты типа *CoordSystem* могут быть созданы, используя:

- код EPSG - European Petroleum Survey Group;
- строку MapInfo PRJ
- строку WKT - Well-known text;
- строку PROJ;
- единиц измерения - для создания СК в план-схеме;

Примечание: Полный список доступных функций создания систем координат содержится в документации к классу *axipy.cs.CoordSystem*.

Например:

```
merc = CoordSystem.from_epsg(3395)
merc.name
```

```
>>> 'Меркатора WGS84'
```

Функция *from_string()* позволяет создавать СК из “универсального представления” - строки с префиксом типа, двоеточием и значением. Возможные префиксы: *proj*, *wkt*, *epsg*, *prj*.

Например:

```
crs = CoordSystem.from_string('prj:Earth Projection 12, 62, "m", 0')
crs.name
```

```
>>> 'Робинсона NAD27'
```

3.1 Трансформация координат

Координаты любой точки земной поверхности в разных системах координат будут различаться, переход от одной системы координат к другой осуществляется с помощью специальных формул преобразований и набора параметров,используемых в этих формулах.

Функционал по переходу координат из одной СК в другую выделен в отельный класс *axipy.cs.CoordTransformer*. Он позволяет преобразовывать координаты между двумя СК.

Например:

```
from axipy import *

transformer = CoordTransformer('epsg:4326', 'epsg:26953')
coordinate = (55.76, 37.6)
result = transformer.transform(coordinate)
f"Point({result.x}, {result.y})"
```

```
>>> 'Point(8513601.095442554, 9873107.576049749)'
```

4

Объекты данных

Разные типы данных обобщены одним абстрактным типом *axipy.da.DataObject*. Он образует иерархию объектов данных различного типа: таблица, растр, грид, чертеж, панорама, и так далее. Объект данных открывается из источника данных *axipy.da.Source* напрямую или неявно.

```
obj = axipy.io.openfile('path/to/mydata.tab')
```

Примечание: При открытии данных они попадают в единый каталог *axipy.da.DataCatalog*.

Для открытия разных источников данных может потребоваться разная информация. Например, для подключения к базе данных нужно указать имя пользователя и пароль и прочее. Поэтому для большинства типов определены функции задания источников данных *axipy.da.DataProvider.get_source()* с дополнительными параметрами. Например, *axipy.da.CsvDataProvider.get_source()*, *axipy.da.PostgreDataProvider.get_source()* и другие.

Например:

```
source = providers.csv.get_source('path/to/mydata.csv', delimiter=';')
table = source.open()
```

Что эквивалентно:

```
table = providers.csv.open('path/to/mydata.csv', delimiter=';')
```

Или:

```
table = providers.openfile('path/to/mydata.csv', delimiter=';')
```

См.также:

Подробнее в документации *axipy.da.DataProviders*.

4.1 Таблицы

Для того, чтобы мы могли работать как с географической, так и с атрибутивной информацией, данные в Аксиоме.ГИС организованы в виде групп файлов, имеющих общее имя, но разные расширения.

Пользователь оперирует только с одним файлом из этой группы – так называемым «табличным» файлом, которые имеет расширение TAB. Все файлы из группы автоматически создаются, обновляются и поддерживаются самой программой Аксиома.ГИС.

4.1.1 Открытие таблиц

Работа с источником данных начинается с открытия объекта данных с помощью функции *openfile()*. Для таблиц возвращаемый объект данных будет типа *axipy.da.Table*.

```
from axipy import io
table = io.openfile('../path/to/datadir/worldcap.tab')
```

Некоторые форматы могут содержать несколько таблиц в одном файле, например GeoPackage. В таком случае нужно указать в параметре `dataobject=`, какую таблицу из файла вы хотите открыть.

```
table = io.openfile('../path/to/datadir/example.gpkg', dataobject='world')
```

4.1.1.1 Источники данных и дополнительные параметры

Источником данных может быть не только файл. Например, это может быть База Данных. Также для открытия или создания некоторых объектов данных может понадобиться указать дополнительные параметры, применимые только для этого типа источника.

Для открытия таких Объектов данных используется функция *open()*, которая принимает словарь со всеми необходимыми параметрами.

Пример открытия таблицы из базы данных PostgreSQL:

```
from axipy import io
definition = {
    "src": "<Адрес сервера БД>",
    "port": 5432,
    "db": "<Имя базы данных>",
    "user": "<Имя прользователя>",
    "password": "<Пароль>",
    "dataobject": '"DataAxi"."World"',
    "provider": "PgDataProvider"
}
table = io.open(definition)
```

Функцию *open()* можно использовать и для рассмотренного нами ранее простого открытия из файла. Можно сказать, что она более универсальна. Функция *openfile()* реализована через функцию *open()*:

```
definition = { 'src': '../path/to/datadir/example.gpkg', 'dataobject': 'world' }
table = io.open(definition)
table.name
```

```
>>> 'world'
```

4.1.1.2 Открытие источников с множеством таблиц

При открытии Таблицы функцией *open()* есть параметр `dataobject`. Он содержит имя таблицы, которую необходимо открыть. Для источников данных с единственной Таблицей этот параметр можно опустить. Для источников с множеством таблиц, например, базы данных или файла GeoPackage, этот параметр обязателен.

Для получения всех доступных таблиц используется функция *read_contents()*:

```
from axipy import io
io.read_contents('../path/to/datadir/example.gpkg')
```

```
>>> ['world', 'worldcap']
```

Для источников с единственной таблицей список будет содержать одно имя:

```
io.read_contents({'src': '../path/to/datadir/worldcap.tab'})
```

```
>>> ['worldcap']
```

4.1.1.3 Провайдеры

Дополнительным параметром при открытии таблиц является Провайдер. Обычно Аксиома может сама определить, каким провайдером открывать файл.

Для получения списка загруженных провайдеров есть функция *loaded_providers()*.

Например:

```
io.loaded_providers()
```

```
{'CsvDataProvider': 'Файловый провайдер: Текст с разделителями',
  'DwgDgnFileProvider': 'Провайдер DWG и DGN (Версия 5.0.19.123)',
  'GdalDataProvider': 'Растровый провайдер GDAL',
  'MifMidDataProvider': 'Провайдер данных MIF-MID',
  'OgrDataProvider': 'Векторный провайдер OGR',
  'PgDataProvider': 'PostgreSQL',
  'RestDataProvider': 'ArcGIS REST',
  'SqliteDataProvider': 'Векторный провайдер sqlite',
  'TabDataProvider': 'MapInfo',
  'TerplanDataProvider': 'Провайдер территориального планирования',
  'TileDataProvider': 'Растровый провайдер тайлов',
  'TmsDataProvider': 'Тайловые сервисы',
  'WfsDataProvider': 'Web Feature Service',
  'WmsDataProvider': 'Web Map Service',
  'WmtsDataProvider': 'Web Map Tile Service',
  'XlsDataProvider': 'Провайдер чтения файлов Excel'}
```

Но иногда провайдер необходимо задать явно. Например, если один тип файлов может быть открыт несколькими провайдерами (например, GeoPackage, который может содержать как таблицу с атрибутикой, так и растр), или при подключении к СУБД.

```
table = io.openfile('../path/to/datadir/example.gpkg', dataobject='worldcap',␣
↪provider='SqliteDataProvider')
table.name
```

```
>>> 'worldcap'
```

У таблицы можно узнать провайдер, которым она была открыта - свойство *axipy.da.DataObject.provider*:

```
table.provider
```

```
>>> 'SqliteDataProvider'
```

4.1.2 Схема таблицы

Записи таблицы имеют фиксированную структуру, повторяющую столбцы таблицы. Схема представлена типом *axipy.da.Schema*. Свойство *axipy.da.Schema.coordsystem* указывает на то, что таблица является пространственной. Атрибуты *axipy.da.Attribute* перечислены в том же порядке, что и столбцы таблицы.

Совет: Схему можно рассматривать как список `list` атрибутов *axipy.da.Attribute*. Манипулировать атрибутами схемы можно также, как элементами списка.

Схему таблицы можно получить методом *axipy.da.Table.schema()*.

Например:

```
schema = table.schema()
```

Схема имеет простую стандартную структуру и с ней просто работать. Например, можно легко вывести все имена атрибутов:

```
schema.attribute_names()
# эквивалентно
[attr.name for attr in schema]
```

4.1.2.1 Атрибуты схемы

Атрибут представлен типом *axipy.da.Attribute*. Его главные параметры - это имя *name* и тип *typedef*. Тип атрибута представляется строкой. В нем может быть указана максимальная длина - для строк и десятичного типа через двоеточие, например, `string:254`. И точность - для десятичного типа через точку, например, `decimal:7.3`.

Доступные типы:

Тип	Описание
string	строка
int	целое число
double	вещественное число с плавающей запятой
decimal	вещественное число с фиксированной запятой
bool	логическое значение
date	дата
time	время
datetime	дата и время

Специальный атрибут Система Координат *coordsystem* содержит значение СК в “универсальном представлении” и указывает на то, что таблица пространственная - может содержать геометрию и стиль.

См.также:

Подробнее в главе *Системы Координат*.

Создание схемы таблицы. Вспомогательные функции

Класс *axipy.da.Attribute* содержит вспомогательные функции `axipy.da.Attribute.string()` и другие для создания атрибутов; а также функцию для создания схемы таблицы - `axipy.da.Attribute.schema()`.

Например, так можно создать схему таблицы:

```
schema = attr.schema(
    attr.string('Столица', 25),
    attr.string('Capital', 25),
    attr.string('Страна', 30),
    attr.string('Country', 30),
    attr.decimal('Cap_Pop', 8, 5),
    coordsystem='prj:Earth Projection 12, 62, "m", 0'
)
```

Свойства *axipy.da.Attribute.length* и *axipy.da.Attribute.precision* позволяют получить длину и точность типа. Список всех свойств описан в справочнике на тип *axipy.da.Attribute*.

4.1.2.2 Чтение записей

Объект таблица *axipy.da.Table* дает возможность работать с записями *axipy.da.Feature*. Метод *axipy.da.Table.items()* возвращает итератор (`iterator`) по записям.

См.также:

Подробнее о записях в главе *Записи*.

```
features = table.items()
for feature in features:
    # обработка записи
    ...
```

Возвращается именно Итератор. При чтении он движется вперед и в конце становится пустым. Чтобы начать чтение сначала, нужно создать новый итератор.

4.1.3 Создание таблиц

Создавать таблицы несколько сложнее, чем открывать готовые. Для них нужно определить схему, СК, Провайдер и пр. Все эти параметры были рассмотрены выше.

Провайдер может быть задан явно:

```
newtable = io.tab.create_open('../path/to/datadir/newtable.tab', schema)
```

или найден автоматически из расширения файла:

```
newtable = io.createfile('../path/to/datadir/newtable.tab', schema)
```

См.также:

axipy.da.DataProviders.tab, *axipy.da.DataProviders.createfile()*.

Добавим в таблицу несколько записей из вселенной Властелин Колец:

```
features_to_insert = [
    Feature({'country': 'Мордор', 'capital': 'Барад-Дур'}),
    Feature(country='Гондор', capital='Минас Тирит'),  # создание с использованием␣
↪**kwargs
    Feature({'country': 'Рохан'}),  # не обязательно подавать все значения, они будут␣
↪пустыми
]
newtable.insert(features_to_insert)
```

4.1.4 Редактирование таблиц

Один из возможных способов редактирования таблиц - открыть исходную таблицу, создать целевую таблицу с той же или другой структурой. И при чтении записей из исходной таблицы редактировать их и записывать в целевую. В результате получится отредактированная копия.

Например, для таблицы `world` необходимо оставить только колонки “Страна” и “Население”; и оставить только те страны, население которых больше `100 миллионов`.

Вот один из вариантов, как это можно реализовать.

```
def is_over_100million(feature) -> bool:
    return feature['Население'] > 100_000_000


orig_table = io.openfile('../path/to/datadir/world.tab')
definition = {
    'src': '../path/to/datadir/edit/edited_world.tab',
    'schema': attr.schema(
        attr.string('Страна'),
        attr.integer('Население'),
    )
}
copy_table = io.create(definition)
orig_features = orig_table.items()

filtered_features = filter(is_over_100million, orig_features)
copy_table.insert(filtered_features)
```

4.1.5 Запросы

SQL-запросы являются мощным инструментом обработки данных. При выполнении запроса к таблицам образуется отдельный объект данных. При открытии данных они попадают в единый каталог *axipy.da.DataCatalog*. Запрос выполняется относительно всех таблиц, находящихся в этом каталоге, с помощью метода *axipy.da.DataCatalog.query()*.

```
query_text = 'SELECT * FROM world WHERE Население > 100000000'
query_table = catalog.query(query_text)
```

4.2 Растры

Растровое изображение – это цифровое представление рисунка, фотографии или иного графического материала в виде набора точек растра.

Класс *axipy.da.Raster* представляет растровый объект данных. Для открытия растровых файлов используйте функцию *openfile()* объекта *axipy.io*.

```
raster = axipy.io.openfile('path/to/raster.tab')
```

4.3 Импорт/Экспорт

4.3.1 Экспорт

Некоторые форматы данных поддерживаются ГИС Аксиомой только на импорт и/или экспорт. Не для всех форматом можно создать, открывать и редактировать данные, используя транзакционную модель редактирования, являющуюся основной для Аксиомы. Тем не менее экспорт и создание-открытие очень близки по назначению, поэтому они объединены и представлены одним типом *axipy.da.Destination* - назначение объекта данных.

Так для некоторых типов экспорт *axipy.da.Destination.export()* является единственной возможностью вывода, в то время как для других - это дополнительная возможность к имеющейся *axipy.da.Destination.create_open()* в случаях, когда открытие и редактирование не требуется.

Экспортировать можно отдельные записи, таблицы или целые источники данных.

Список 1: Пример экспорта таблицы

```
destination = io.csv.get_destination(output_filepath, Schema())
destination.export_from_table(table, copy_schema=True)
```

4.3.2 Импорт

Источник данных *axipy.da.Source* - это зеркальный тип назначения объекта данных *axipy.da.Destination*. Так же как для назначения, некоторые типы данных поддерживают только импорт, и не могут быть напрямую открыты с помощью *axipy.da.Source.open*. А другие типы поддерживают и открытие и импорт для случаев, когда открытие и редактирование не требуется.

Список 2: Пример экспорта источника

```
source = io.tab.get_source(input_tabfile)
destination.export_from(source)
```

См.также:

Чтобы узнать, какие типы поддерживают импорт и экспорт, обратитесь к описанию конкретных провайдеров данных *axipy.da.DataProviders*.

5 Записи

Запись *Feature*, которая получается при чтении из таблицы, во многом повторяет словарь Python `dict`. В ней содержатся пары (Имя столбца: Значение). Причем значение приводится к типу столбца. Например, если это числовое поле int, а его значение 42, то запись будет содержать число 42, а не строку “42”.

Прочитанная запись никак не ссылается на таблицу и является копией. Присвоенная к переменной запись будет доступна после продвижения итератора или даже после закрытия таблицы. Но поэтому и изменения, внесенные в эту запись-копию, никак не повлияют на значения в таблице.

5.1 Атрибуты

Доступ атрибутам осуществляется по имени или номеру. Так же как для словаря, если атрибут с заданным именем не существует, то вызывается исключение `KeyError`. Если атрибут с заданным номером не существует, то вызывается исключение `IndexError`.

```
feature = next(table.items())
try:
    value = feature['attr_name']
except KeyError as e:
    print(f'Поймано исключение: {e}')
```

```
>>> Поймано исключение: "Key 'unknown_key' not found"
```

Можно задать значение по-умолчанию при чтении атрибута, который может не существовать. Если его не задать, то значение по-умолчанию считается равным `None`.

```
value = feature.get('attr_name', 0.0)
```

Проверка существования атрибута с помощью ключевого слова `in` так же, как в словаре:

```
if 'attr_name' in feature:
    ...
```

5.1.1 Геометрический атрибут

Доступ к специальному атрибуту Геометрия *axipy.da.Geometry* производится через свойство *axipy.da.Feature.geometry*.

Или можно использовать специальное наименование GEOMETRY_ATTR, представляющее имя геометрического атрибута:

```
geometry = feature.geometry
# эквивалентно
geom = feature[GEOMETRY_ATTR]
```

Проверка существования *has_geometry()*:

```
if feature.has_geometry():
    ...
# эквивалентно
if GEOMETRY_ATTR in feature:
    ...
```

5.1.2 Стиль для геометрического атрибута

Стиль *axipy.da.Style* может содержаться в виде атрибута. Доступ к нему производится по специальному наименованию `STYLE_ATTR или свойствам *axipy.da.Feature.style* и *has_style()*.

```
style = feature.style
# эквивалентно
style = feature[STYLE_ATTR]
```

```
feature.has_style()
# эквивалентно
STYLE_ATTR in feature
```

5.2 Идентификаторы записей

Записи имеют свойство *axipy.da.Feature.id*. Это значение зависит от типа данных. При этом не гарантируется порядок, начальное значение или отсутствие разрывов между соседними записями. Также идентификатор необязательно является числом.

6 Геометрия

Геометрический объект (геометрия) представляет собой описательную структуру данных, на базе которой формируется графическое представление векторного элемента. Оно может быть частью визуального представления объектов реального мира. В зависимости от целей, геометрия может содержать данные о системе координат, в которой она создана.

6.1 Типы

Аксиома поддерживает следующие типы геометрических объектов:

Простые типы геометрии:

- *Точка*
- *Полилиния*
- *Полигон*

Коллекции:

- *Смешанная коллекция*
- *Коллекция точек*
- *Коллекция полилиний*
- *Коллекция полигонов*

Так же возможна работа с типами данных MapInfo:

- *Линия*
- *Прямоугольник*
- *Скругленный прямоугольник*
- *Эллипс*
- *Дуга*
- *Текст*

Рассмотрим подробнее геометрические типы. Переменные из примеров создания объектов будут использованы ниже в примерах более общего характера. Более общие характеристики объектов, а так же операции над ними будут рассмотрены ниже.

6.1.1 Точка

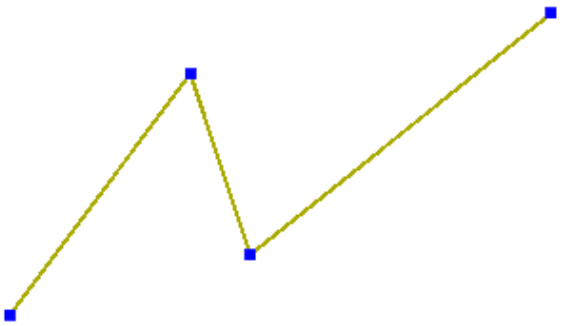
Точка представлена классом *axipy.da.Point*. Для нее характерно отсутствие таких параметров, как площади и линейных размеров. Создадим точку с координатами (10, 10) в СК Широта/Долгота. С целью визуального восприятия, результат представим в виде WKT строки (*axipy.da.Geometry.wkt*).

```
from axipy import *
cs = CoordSystem.from_prj("1, 104")
point = Point(10, 10, cs)
print('Точка ({}, {})'.format(point.x, point.y))

>>> Точка (10, 10)
```

6.1.2 Полилиния

Представлена классом *axipy.da.LineString* в виде непрерывной линии, соединяющей последовательность узлов. Для нее так же характерно отсутствие площади, но присутствует длина. Точки полилинии хранятся в виде списка `list` [*axipy.utl.Pnt*], то при задании, помимо передачи в конструктор такого списка, так же допустимо указание координат в виде пар координат `tuple`. Нумерация точек при доступе по индексу начинается с 0. Доступны через свойство *axipy.da.LineString.points* Приведем пример: создадим полилинию без СК. В этом же примере заменим 3-ю вершину на другое значение и выведем полученный результат в виде wkt *axipy.da.Geometry.wkt*:

```
ls = LineString([(1, 1), (4, 5), (5, 2), (10, 6)])
ls.points[2] = (6, 3)
print(ls.wkt)

>>> LINESTRING (1 1, 4 5, 6 3, 10 6)
```



Так же присутствует возможность изменения существующих параметров полилинии. Добавим точку в позицию 1 и удалим точку 2:

```
ls.points.insert(1, (3,6))
ls.points.remove(2)
print(wkt)

>>> LINESTRING (1 1, 3 6, 6 3, 10 6)
```

Поддерживается инициализация через существующий итератор. Смоделируем данную ситуацию: создадим итератор на базе точек первой полилинии и на его основе создадим полилинию:

```
itr = (a for a in ls.points)
ls_it = LineString(itr)
```

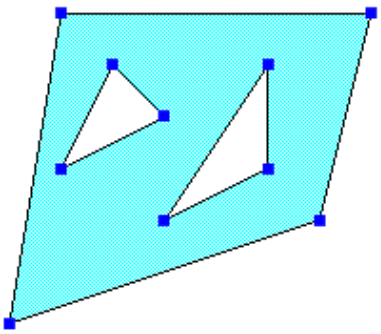
6.1.3 Полигон

Представлен классом *axipy.da.Polygon*. Полигон представляет собой площадной объект, или другими словами часть плоскости, ограниченная замкнутой полилинией. Кроме внешней границы, полигон может иметь одну или несколько внутренних (дырок). Ему присущи такие свойства, как длина (периметр) и площадь. Точки хранятся по аналогии с полилинией. И инициализация в конструкторе или при добавлении дырки в полигон производится по подобному принципу. Характерной особенностью является тот факт, что все контуры замкнуты и последняя точка совпадает с первой. Это касается как формы полигона, так и его дырок. В качестве примера создадим полигон:

```
polygon = Polygon((1,1), (2,7), (8,7), (7,3))
print(polygon.wkt)

>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1))
```

И добавим в него две дырки:

```
polygon.holes.append((2,4), (3,6), (4,5))
polygon.holes.append((4,3), (6,6), (6,4))
print(polygon.wkt)

>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 4, 4 3))
```



Как уже указывалось выше, работа с точками для полигона производится аналогично с полилинией. Это же касается и дырок, только доступ производится через свойство *axipy.da.Polygon.holes* Обновим значение третьей точки для второй дырки.

```
polygon.holes[1][2] = (6,3)
print(polygon.wkt)

>>> POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 3, 4 3))
```

6.1.4 Смешанная коллекция

Представлена классом *axipy.da.Collection*. Это нетипизированная коллекция. Может содержать внутри себя геометрии различных типов за исключением коллекций. Попробуем создать коллекцию и добавить в нее последовательно точку, полилинию и полигон. Стоит обратить внимание, что если добавляется последовательность точек, то она рассматривается как полилиния, в тоже время для указания полигона необходимо явно указать принадлежность к классу. Доступ к элементам коллекции производится по индексу, начиная со значения 0.

```
coll = Collection()
coll.append(1,2) # Точка
coll.append((3,4), (5, 5), (10, 0)) # Полилиния
coll.append(Polygon((3,4), (5, 5), (10, 0))) # Полигон
print(coll.wkt)

>>> GEOMETRYCOLLECTION (POINT (1 2), LINESTRING (3 4, 5 5, 10 0), POLYGON ((3 4, 5 5,␣
↪10 0, 3 4)))
```

Удалим из коллекции полилинию и полигон, а после этого добавим объекты, созданные в предыдущих примерах. Точку поменяем простой заменой по индексу:

```
coll.remove(2)
coll.remove(1)
coll.append(polygon)
coll.append(ls)
coll[0] = point
print(coll.wkt)

>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6,␣
↪4 5, 2 4), (4 3, 6 6, 6 3, 4 3)), LINESTRING (1 1, 3 6, 6 3, 10 6))
```

При замене элемента с заданием координат работают такие же принципы, как и в конструкторе. Обновим полилинию и полигон:

```
coll[2] = [(101, 102), (103, 104), (105, 106)]
coll[1] = Polygon((101, 102), (103, 104), (105, 106))
print(coll.wkt)

>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((101 102, 103 104, 105 106, 101 102)),
↪ LINESTRING (101 102, 103 104, 105 106))
```

Поменяем первую (она же последняя) точку полигона:

```
coll[1].points[0] = (0,0)
print(coll.wkt)

>>> GEOMETRYCOLLECTION (POINT (10 10), POLYGON ((0 0, 103 104, 105 106, 0 0)),␣
↪LINESTRING (101 102, 103 104, 105 106))
```

Далее рассмотрим типизированные коллекции. Принципы работы аналогичны нетипизированным, за исключением того, что позволено хранение геометрий только одного типа.

6.1.5 Коллекция точек

Представлена классом *axipy.da.MultiPoint*. Это типизированная коллекция. Допустимо хранение только точек. Создадим коллекцию точек и добавим в нее 2 элемента разными способами:

```
mpoint = MultiPoint()
mpoint.append(10,10)
mpoint.append((12, 12))
print(mpoint.wkt)

>>> MULTIPOINT (10 10, 12 12)
```

6.1.6 Коллекция полилиний

Представлена классом *axipy.da.MultiLineString*. Это типизированная коллекция. Допустимо хранение только полилиний.

```
mls = MultiLineString() # Создадим саму коллекцию.
mls.append((11, 12), (13, 14), (15, 16)) # Добавим как объект
mls.append([(21, 22), (23, 24), (25, 26)]) # Добавим как перечень точек
print(mls.wkt)

>>> MULTILINESTRING ((11 12, 13 14, 15 16), (21 22, 23 24, 25 26))
```

6.1.7 Коллекция полигонов

Представлена классом *axipy.da.MultiPolygon*. Это типизированная коллекция. Допустимо хранение только полигонов. Отдельно стоит отметить, что при добавлении/изменения геометрий в виде перечня точек, в отличие от смешанной коллекции и коллекции полилиний, в данном случае создается полигон. Рассмотрим на примере:

```
mpoly = MultiPolygon()
mpoly.append((1, 2), (3, 4), (5, 6), (7, 8))
mpoly.append(poly) # Добавим ранее созданный с дыркой
print(mpoly.wkt)

>>> MULTIPOLYGON (((1 2, 3 4, 5 6, 7 8, 1 2)), ((0 0, 1 10, 14 15, 11 5, 10 2, 0 0),␣
↪(2 2, 44 44, 5 3, 2 2), (2 2, 2 4, 5 3, 2 2)))
```

Далее рассмотрим объекты MapInfo. Одна из особенностей заключается в том, что они нестандартные, и, как следствие, не поддерживают WKT представление.

6.1.8 Линия

Представлена классом *axipy.da.Line*. Линейный объект. Описывается двумя точками: начальным и конечным узлом. Создадим линию, передав в конструктор пару точек. После этого последовательно поменяем координаты начала и конца линии.

```
line = Line((11, 11), (21, 21))
line.begin = (12, 12)
line.end = (120, 120)
```

6.1.9 Прямоугольник

Представлен классом *axipy.mi.Rectangle*. Площадной объект. Описывается минимальными и максимальными значениями по координатам X и Y. Создадим прямоугольник, задав параметры через конструктор. Затем поменяем предельные значения по X координате. Результат проконтролируем выводом значения *axipy.da.Geometry.bounds* геометрии.

```
rectangle = Rectangle(0, 0, 40, 20)
rectangle.xmin = 10
rectangle.xmax = 50
print(rectangle.bounds)

>>> (10.0 0.0) (50.0 20.0)
```

6.1.10 Скругленный прямоугольник

Представлен классом *axipy.mi.RoundRectangle*. Так же является площадным объектом. Отличительной особенностью от прямоугольника является наличие скруглений на углах. В остальном данные объекты аналогичны. Во избежании путаницы с параметрами, в конструкторе задается *axipy.utl.Rect* и радиусы скругления:

```
rrectangle = RoundRectangle([0, 0, 40, 20], 0.2, 0.2)
rrectangle.xRadius = 0.3
```

6.1.11 Эллипс

Представлен классом *axipy.mi.Ellipse*. Является площадным объектом. Создадим эллипс, передав в конструктор его Rect. После этого поменяем свойства.

```
ellipse = Ellipse([0,0,22,33])
ellipse.center = (10,10) # Переопределим центр
ellipse.majorSemiAxis = 10 # Задание большой полуоси
ellipse.minorSemiAxis = 5 # Задание малой полуоси
print(ellipse.bounds)

>>> (0.0 5.0) (20.0 15.0)
```

6.1.12 Дуга

Представлена классом *axipy.mi.Arc*. Линейный объект. Задается в конструкторе через *axipy.utl.Rect* и, дополнительно, начальный и конечный угол дуги. Создадим объект, затем выведем основные свойства:

```
arc = Arc(Rect(0,0,20,30), 45, 270)
print('center={} start={} end={}'.format(arc.center, arc.startAngle, arc.endAngle))

>>> center=(10.0 15.0) start=45.0 end=270.0
```

6.1.13 Текст

Представлен классом *axipy.mi.Text*. В отличие от описанных выше типов, его геометрические свойства определяются не только параметрами класса, но и параметрами его оформления.

```
text = Text("Пример", (10, 10))
```

Рассмотрим общие свойства геометрии.

6.2 Геометрические свойства

В зависимости от типа объекта, для него могут быть получены свойства. Продемонстрируем использование некоторых из них:

```
polygon = Polygon((0, 0), (0, 10), (10, 10), (10, 0))
print(f'Название: {polygon.name}')
print(f'Площадь: {polygon.area}')
print(f'Периметр: {polygon.length}')
print(f'Ограничивающий прямоугольник: {polygon.bounds}')
```

```
>>> Название: Полигон
>>> Площадь: 100.0
>>> Периметр: 40.0
>>> Ограничивающий прямоугольник: (0.0 0.0) (10.0 10.0)
```

6.2.1 Сериализация

Аксиома позволяет производить сериализацию геометрию в форматы текстового WKT или бинарного вида WKB. Подробнее

Рассмотрим на примере точечного объекта сначала для случая WKT. Будем использовать метод *axipy.da.Geometry.from_wkt()* для получения объекта из WKT и свойство *axipy.da.Geometry.wkt* для получения WKT представления полученного объекта:

```
pnt = Geometry.from_wkt('POINT (10 10)')
print(isinstance(pnt, Point))
print('wkt {}'.format(pnt.wkt))
```

```
>>> True
>>> wkt POINT (10 10)
```

6.3 Преобразования

Для перепроецирование в другую СК используется метод *axipy.da.Geometry.reproject()*. Заметим, что исходный объект должен содержать свою СК. В рамках примера перепроецируем точку из СК Широта/Долгота в проекцию Меркатора:

```
csLL = CoordSystem.from_prj("1, 104")
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
point_ll = Point(10,10, csLL)
point_merc = point_ll.reproject(csMercator)
print('point result: {}'.format(point_merc.wkt))

>>> point result: POINT (1113194.90793274 1111475.10285222)
```

Более простые преобразования типа сдвига *axipy.da.Geometry.shift()*, масштабирования *axipy.da.Geometry.scale()* и поворот *axipy.da.Geometry.scale()* игнорируют указанную СК и данные операции производятся в текущих координатах объекта. Рассмотрим на примерах:

```
polygon = Polygon((1,1), (2,7), (8,7), (7,3))
polygon_shift = polygon.shift(5,5)
# Сдвинем на величину 5 по X и Y
print('polygon shift: {}'.format(polygon_shift.wkt))
polygon_scale = polygon.scale(5,5)
# Отмасштабируем координаты относительно центра
print('polygon scale: {}'.format(polygon_scale.wkt))
# Повернем на 90 градусов против часовой стрелки относительно точки (10, 40)
polygon_rotated = polygon.rotate((10, 40), 90)
print('polygon rotate: {}'.format(polygon_rotated.wkt))
```

```
>>> polygon shift: POLYGON ((6 6, 7 12, 13 12, 12 8, 6 6))
>>> polygon scale: POLYGON ((-13 -11, -8 19, 22 19, 17 -1, -13 -11))
>>> polygon rotate: POLYGON ((49 31, 43 32, 43 38, 47 37, 49 31))
```

Для более сложных аффинных преобразований можно использовать *axipy.da.Geometry.affine_transform()* с указанием матрицы преобразований `QTransform`

6.4 Пространственные операции

6.4.1 Нормализация объекта

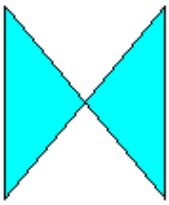
Для проверки валидности геометрии предусмотрено свойство *axipy.da.Geometry.is_valid*. Если геометрия не проходит тест на валидность (данное свойство False), то для его нормализации используется метод `axipy.da.Geometry.normalize()`. Краткую аннотацию причины почему геометрия недействительна, можно воспользовавшись свойством *axipy.da.Geometry.is_valid_reason*.

Создадим заведомо неправильный полигон и попробуем его исправить:

```
poly_bad = Polygon([(1, 1), (6, 7), (6, 1), (1, 7)])
poly_norm = poly_bad.normalize()
print('Validate source: {} ({})'.format(poly_bad.is_valid, poly_bad.is_valid_reason))
print('Validate destination: {}'.format(poly_norm.is_valid))
print('Wkt:{}'.format(poly_norm.wkt))
```

```
>>> Validate source: False (Self-intersection[3.5 4])
>>> Validate destination: True
>>> Wkt:MULTIPOLYGON (((3.5 4, 1 1, 1 7, 3.5 4)), ((3.5 4, 6 7, 6 1, 3.5 4)))
```

В результате мы получили коллекцию из двух полигонов.



6.4.2 Клонирование объекта

Для создания копии объекта используется метод *axipy.da.Geometry.clone()*:

```
point1 = Point(10, 10)
point2 = point1.clone()
point1.x = 12
print('>>>', point1.wkt, point2.wkt)
```

```
>>> POINT (12 10) POINT (10 10)
```

6.4.3 Логические операции

Пространственные отношения, возвращающие логический `True` или `False`:

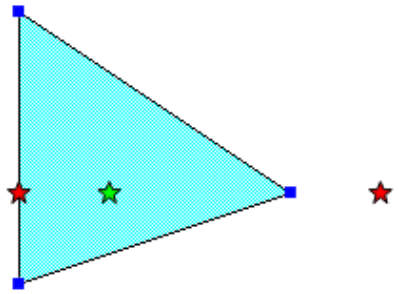
Точное совпадение геометрий производится посредством *axipy.da.Geometry.equals()*, если же необходимо произвести приблизительное сравнение, то используем метод *axipy.da.Geometry.almost_equals()*:

```
polygon1 = Polygon((1, 1), (2, 7), (7, 3))
polygon2 = Polygon((1, 1.1), (2, 7), (7, 3))
print('Точное сравнение:', polygon1.equals(polygon2))
print('Сравнение с точностью 0.2:',polygon1.almost_equals(polygon2, 0.2))
```

```
>>> Точное сравнение: False
>>> Сравнение с точностью 0.2: True
```

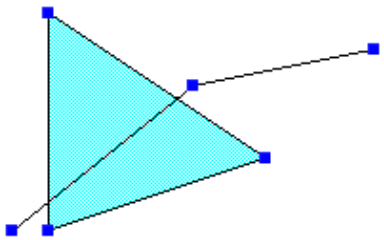
Проверка на попадание *axipy.da.Geometry.contains()*

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
point1 = Point(3,3)
point2 = Point(9,3)
point3 = Point(1,3)
print('Точка внутри:', poly1.contains(point1))
print('Точка снаружи:', poly1.contains(point2))
print('Точка на грани:', poly1.contains(point3))

>>> Точка внутри: True
>>> Точка снаружи: False
>>> Точка на грани: False
```



Проверка на частичное пересечение *axipy.da.Geometry.crosses()*

```
sl = LineString ((0, 1), (5, 5),  (10, 6))
print(poly1.crosses(sl))

>>> True
```



Проверка на отсутствие соприкосновений *axipy.da.Geometry.disjoint()*

```
print(poly1.disjoint(sl))

>>> False
```

Проверка пересечений объектов *axipy.da.Geometry.intersects()*

Пересечение геометрий, если результат отличен от анализируемых данных *axipy.da.Geometry.overlaps()*

```
poly2 = Polygon((5,1), (4,4), (10,3))
print(poly1.overlaps(poly2))

>>> True
```

Проверка касания *axipy.da.Geometry.touches()*

```
print('Точка на грани:', poly1.touches(point3))

>>> True
```

Функция, обратная *contains* *axipy.da.Geometry.within()*

```
print('Точка внутри:', point1.within(poly1))

>>> True
```

axipy.da.Geometry.covers()

6.4.4 Отношения DE-9IM

Функция *axipy.da.Geometry.relate()* проверяет все DE-9IM отношения между объектами. Предикаты выше являются их частными случаями.

```
print('relate1:', poly1.relate(point1))
print('relate2:', poly1.relate(point2))
```

```
>>> relate1: 0F2FF1FF2
>>> relate2: FF2FF10F2
```

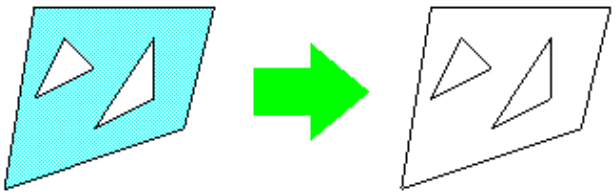
6.4.5 Операции над объектами

В данном разделе рассмотрим операции, результатом выполнения которых будут новые объекты.

Получение границ геометрии в виде полилинии *axipy.da.Geometry.boundary()*

```
poly = Geometry.from_wkt('POLYGON ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4),␣
↪(4 3, 6 6, 6 4, 4 3))')
poly_boundary = poly.boundary()
print(poly_boundary.wkt)
```

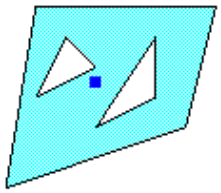
```
>>> MULTILINESTRING ((1 1, 2 7, 8 7, 7 3, 1 1), (2 4, 3 6, 4 5, 2 4), (4 3, 6 6, 6 4,␣
↪4 3))
```



Центроид объекта *axipy.da.Geometry.centroid()*

```
centroid = poly.centroid()
print(centroid.wkt)
```

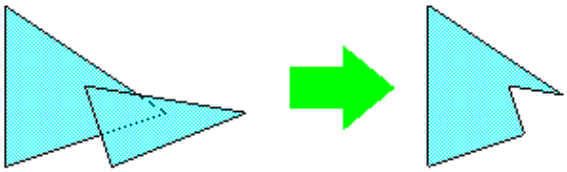
```
>>> POINT (4 4.5)
```



Вычитание объектов *axipy.da.Geometry.difference()*

```
print(poly1.difference(poly2).wkt)
```

```
>>> POLYGON ((1 1, 1 7, 6 3.67, 4 4, 4.6 2.2, 1 1))
```



Пересечение объектов *axipy.da.Geometry.intersection()*

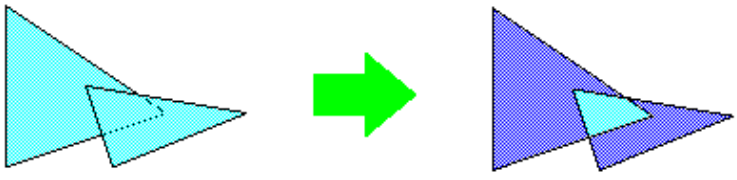
```
print(poly1.intersection(poly2).wkt)
```

```
>>> POLYGON ((6 3.67, 7 3, 4.6 2.2, 4 4, 6 3.67))
```

Обратное пересечение объектов *axipy.da.Geometry.symmetric_difference()*

```
print(poly1.symmetric_difference(poly2).wkt)
```

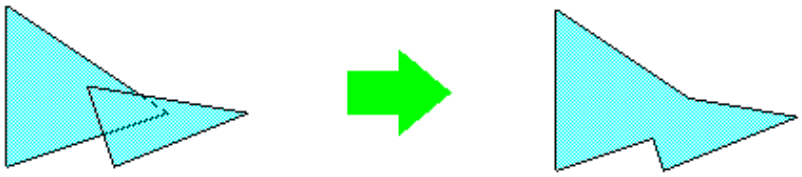
```
>>> MULTIPOLYGON (((1 1, 1 7, 6 3.67, 4 4, 4.6 2.2, 1 1)), ((4.6 2.2, 7 3, 6 3.67, 10␣
↪3, 5 1, 4.6 2.2)))
```



Объединение объектов *axipy.da.Geometry.union()*

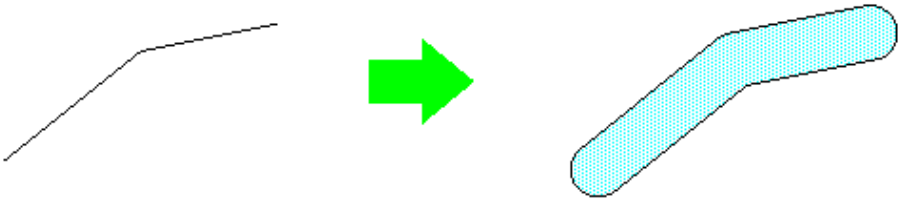
```
print(poly1.union(poly2).wkt)
```

```
>>> POLYGON ((1 1, 1 7, 6 3.67, 10 3, 5 1, 4.6 2.2, 1 1))
```



Построение буфера *axipy.da.Geometry.buffer()*

```
buf = sl.buffer(1)
```

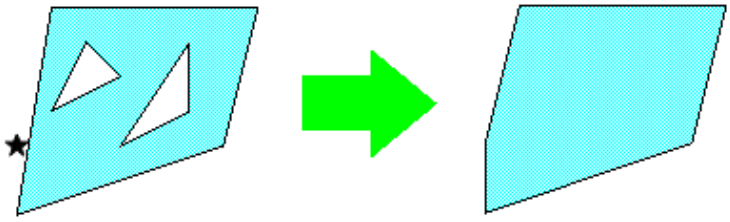


Границы объекта *axipy.da.Geometry.convex_hull()*

Рассмотрим на примере коллекции:

```
coll = Collection()
coll.append(poly)
coll.append(point3)
print(coll.convex_hull().wkt)
```

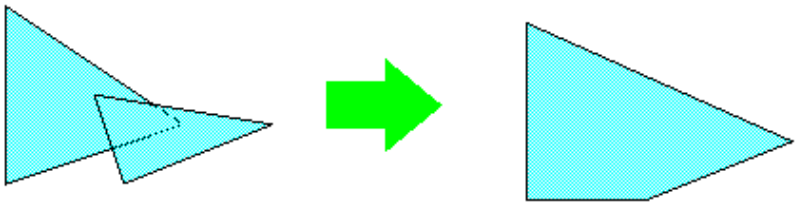
```
>>> POLYGON ((1 1, 1 3, 2 7, 8 7, 7 3, 1 1))
```



Пример с внутренними углами:

```
print(poly1.union(poly2).convex_hull().wkt)
```

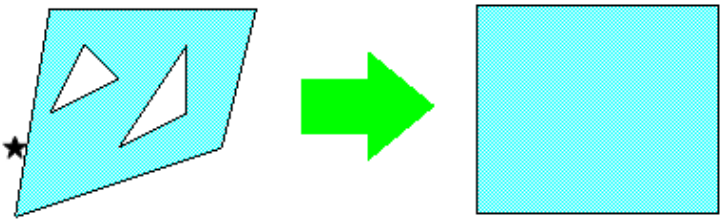
```
>>> POLYGON ((1 1, 1 7, 10 3, 5 1, 1 1))
```



Прямоугольные границы объекта *axipy.da.Geometry.envelope()*

```
print(coll.envelope().wkt)
```

```
>>> POLYGON ((1 1, 8 1, 8 7, 1 7, 1 1))
```



7 Стиль

Стиль представляет собой описательную структуру, которая в свою очередь используется при оформлении геометрического объекта `Geometry` при его отрисовке. Стиль представлен базовым классом *`axipy.da.Style`*, а так-же его наследниками.

При работе с табличными данными стиль, при наличии в ней геометрического атрибута, может определяться тремя разными способами:

- Содержаться в специальной колонке таблицы в виде атрибута. В данном случае для геометрии каждой записи таблицы назначается соответствующий ей стиль.
- Определяется для колонки на уровне таблицы. В данном случае геометрия всех записей будет иметь одинаковое оформление.
- В таблице присутствует только геометрия. В данном случае стиль при отрисовке слоя будет браться как значение по умолчанию.

Рассмотрим в дальнейшем первый вариант. Для выполнения последующих примеров создадим таблицу в памяти и зарегистрируем ее в системе:

```
definition = {
  'src': '',
  'schema': attr.schema(
    attr.string('id', 60),
    coordsystem='prj:1, 104'
  )
}
table = io.create(definition)
app.mainwindow.catalog().add(table)
```

Далее попробуем различными методами добавить геометрию в эту таблицу. На примере точечного объекта. Создадим точечный объект, и, если стиль оформления не имеет значения, назначим ему наиболее подходящий для данного типа (в нашем случае точки) объекта, просто передав туда нашу геометрию:

```
point = Point(10,8)
pstyle = Style.for_geometry(point)
print(pstyle.to_mapinfo())

>>> Symbol (36, 255, 12, "Map Symbols", 0,0)
```

Для иллюстрации результата сформируем на базе созданных объектов геометрии и стиля запись и добавим его в ранее созданную таблицу. Итог покажем на карте:

```
fpoint = Feature(
  geometry=point,
  style=pstyle
)
table.insert([fpoint])
m = Map([table])
view = view_service.create_mapview(m)
window = app.mainwindow.add(view)
window.show()
```

В результате получим:

Так же доступно создание из строки формата MapBasic. Для этого используется метод *axipy.da.Style.from_mapinfo()*. Если же для существующего стиля необходимо получить строку MapBasic, то используется метод *axipy.da.Style.to_mapinfo()*. Создадим для нашей точки стиль на базе представления MapBasic. Строка формирования стиля точки будет выглядеть так:

```
pstyle = Style.from_mapinfo('Symbol (35, 255, 20)')
```

Пример добавления точки, но с использованием стиля, на основании растрового символа:

```
pstyle = PointStyle.create_mi_picture("GLOB1-32.bmp")
print(pstyle)

fpoint = Feature(
  geometry=point,
  style=pstyle
)
```

```
>>> Symbol ("GLOB1-32.bmp", 0, 12, 0)
```

Далее вставим в таблицу по аналогии, рассмотренной выше. Результат:

Теперь рассмотрим объект типа полигон.

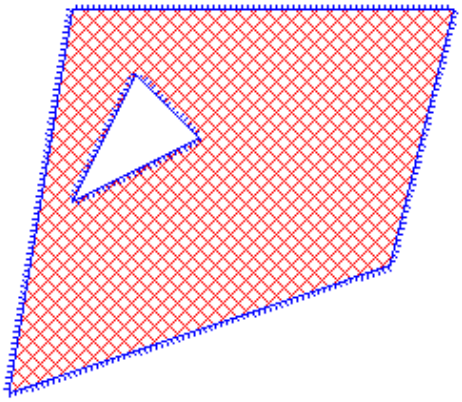
```
from PySide2.QtCore import Qt

polygon = Polygon((1,1), (2,7), (8,7), (7,3))
polygon.holes.append((2,4), (3,6), (4,5))
polystyle = PolygonStyle()
polystyle.set_pen(pattern=48, color=Qt.blue)
polystyle.set_brush(pattern=8, color=Qt.red)
print(polystyle)
fpolygon = Feature(
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  geometry=polygon,
  style=polystyle
)
table.insert([fpolygon])
```

```
>>> Pen (1, 48, 255)  Brush (8, 16711680, 0)
```

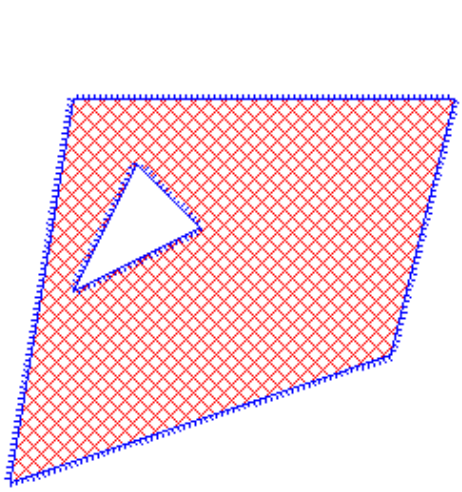


Для сложных разнородных объектов применяются стили, которые внутри себя содержат другие стили для каждого типа используемых объектов. Для этого используется класс *axipy.da.CollectionStyle*. Из ранее созданных полигона и точки сделаем коллекцию посредством объединения. И, одновременно с этим, на базе ранее созданных стилей сделаем сложный стиль:

```
collstyle = CollectionStyle()
collstyle.for_point(pstyle)
collstyle.for_polygon(polystyle)
print(collstyle)
collection = polygon.union(point)
fcollection = Feature(
  geometry=collection,
  style=collstyle
)
table.insert([fcollection])
```

```
>>>  Symbol ("GLOB1-32.bmp", 0, 12, 0) Pen (1, 48, 255)  Brush (8, 16711680, 0)
```

В результате получим следующий объект:



8 Отображение данных

8.1 Слой

Для отображения данных таблицы или растра необходимо создать на основе этого источника данных слой *axipy.render.Layer*.

```
from axipy.render import Layer

layer = Layer.create(table)
```

В зависимости от типа передаваемого объекта будет создан векторный или растровый слой.

8.2 Карта

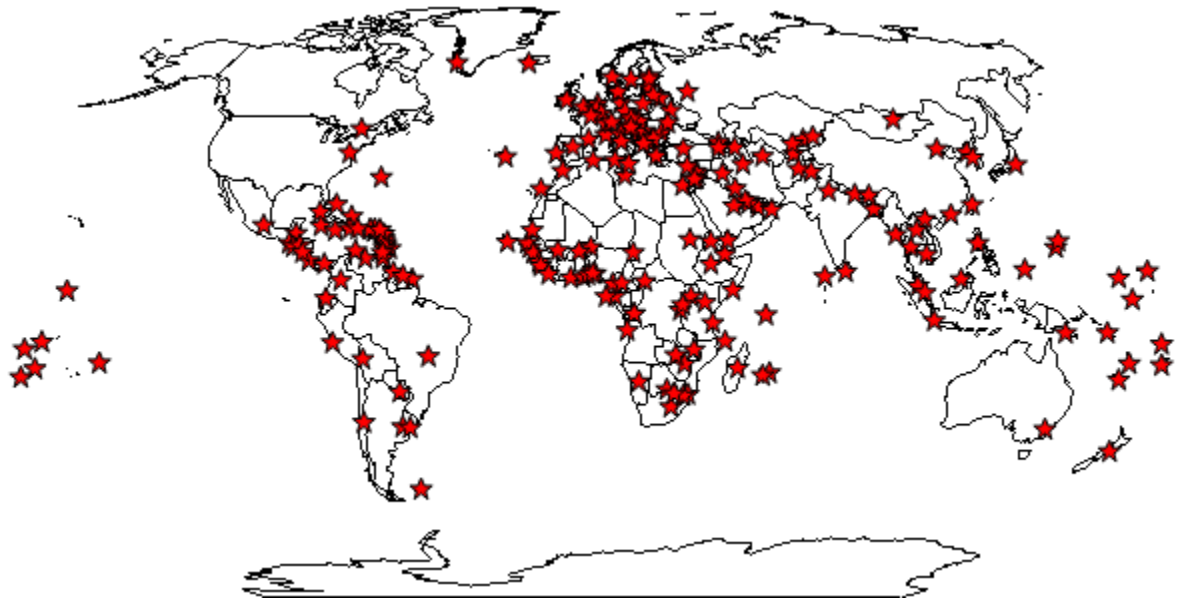
Совокупность слоев образует карту *axipy.render.Map*. Порядок отрисовки слоев прямо зависит от их расположения в карте. То есть первый слой будет на самом верху, а последний - в самом низу.

Создадим карту с двумя слоями. Передадим в карту список таблиц - из них автоматически создадутся слои с параметрами по умолчанию.

```
from axipy import io
from axipy.render import Map


world = io.openfile('../path/to/datadir/example.gpkg', dataobject='world')
capital = io.openfile('../path/to/datadir/worldcap.tab')
map = Map([capital, world])
```

Карту можно вывести в изображение - PySide2.QtGui.QImage, которое, например, можно в качестве результата сохранить в файл.

```
image = map.to_image(600, 300)
```



Полученную картинку можно сохранить как растр в файловой системе. Формат файла будет определяться его расширением:

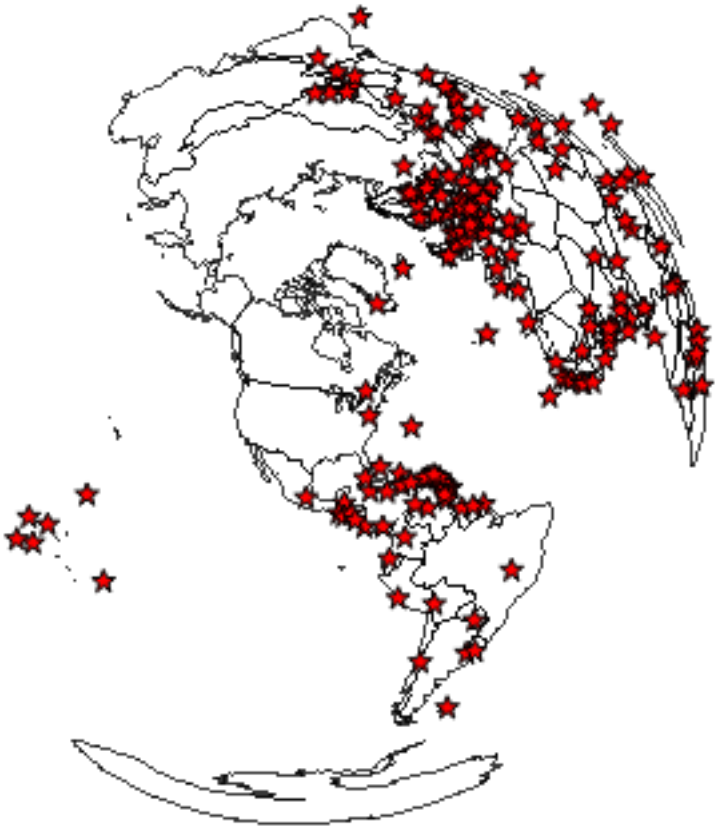
```
image.save('../path/to/outdir/map.png')
```

```
>>> True
```

Мы указали только размеры изображения. Карта вывелась в Системе Координат (СК), наиболее подходящей для отображения слоев в ней. В нашем случае обе таблицы оказались в одной СК, которая и была выбрана для отображения.

Теперь отрисуем эту же карту Азимутальной СК:

```
from axipy.cs import CoordSystem

azimuth = CoordSystem.from_epsg(2163)
image = map.to_image(600, 300, coordsystem=azimuth)
```



Границы карты по умолчанию определились равными границам СК. Снова нарисуем нашу карту уже в Широте/Долготе в границах, примерно включающих Италию. Ограничивающий прямоугольник указываем в градусах:

```
longlat = CoordSystem.from_epsg(4326)
image = map.to_image(600, 300, coordsystem=longlat, bbox=(5, 35, 15, 15))
```

Теперь попробуем для слоя *capital* задать выражение для отображаемых меток *axipy.render.VectorLayer.label*, и для обоих слоев переопределить *стиль оформления*, сделав его однообразным *axipy.render.VectorLayer.overrideStyle*. Заметим, что перечень доступных слоев карты доступен через свойство *axipy.render.Map.layers*. Т.е. помимо передачи перечня слоев в конструктор карты, также возможно управление этим списком позже.

```
from axipy.da import *

lay_capital = map.layers[0]
lay_capital.label.text = 'Столица'
lay_capital.label.placementPolicy = Label.DISALLOW_OVERLAP
lay_capital.overrideStyle = Style.from_mapinfo('Symbol (34,255,6)')
lay_world = map.layers['world']
lay_world.overrideStyle = Style.from_mapinfo('Pen (1, 2, 0) Brush (8, 255)')
image = map.to_image(600, 300, coordsystem=longlat, bbox=(5, 35, 15, 15))
```

В рамках примера по управлению слоями в конце удалим слой со столицами (самый верхний):

```
map.layers.remove(0)
```

Создадим карту на базе растра как подложки и установим прозрачным цвет фона.

```
from axipy import *
from PySide2.QtGui import QColor

raster = io.openfile('../path/to/datadir/TrueMarble.tab')
rasterLayer = Layer.create(raster)
rasterLayer.transparentColor = QColor('#000014')

mapRaster = Map([capital, rasterLayer])
image = mapRaster.to_image(600, 320)
```

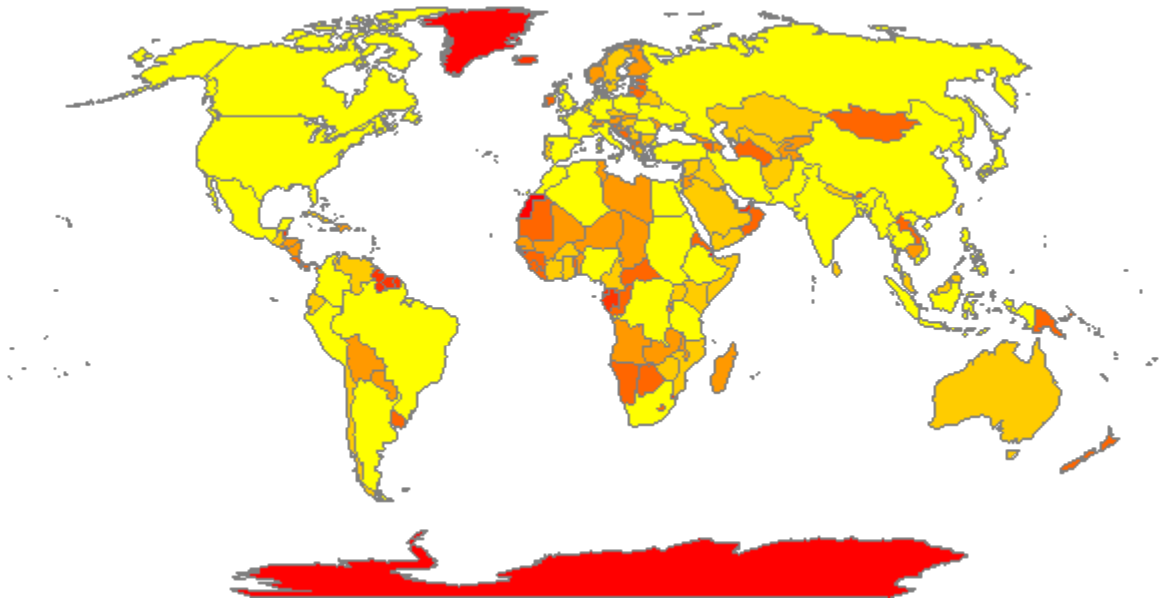
8.3 Тематические слои

Тематическая карта отображает ваши данные в виде условных знаков, выделяя их оттенками, цветами, штриховками, а также представляя их в виде столбчатых и круговых диаграмм.

Для векторных слоев *axipy.render.VectorLayer* есть возможность формирования и отрисовки тематических слоев. Т.е. применить оформление на базе атрибутивной информации. Рассмотрим на примере тематики по интервалам. Построим тематику по атрибутивному полю “Население” на 6 интервалов с равномерным распределением по количеству записей. Цвета распределим градиентом от желтого до красного.

Тематические слои добавляются как дочерние к их базовому слою.

```
from axipy.render import RangeThematicLayer
from PySide2.QtCore import Qt


world_layer = map.layers[0]
thematic = RangeThematicLayer('Население')
thematic.ranges = 6
thematic.assign_two_colors(Qt.red, Qt.yellow)
thematic.splitType = RangeThematicLayer.EQUAL_COUNT
world_layer.thematic.add(thematic)
```



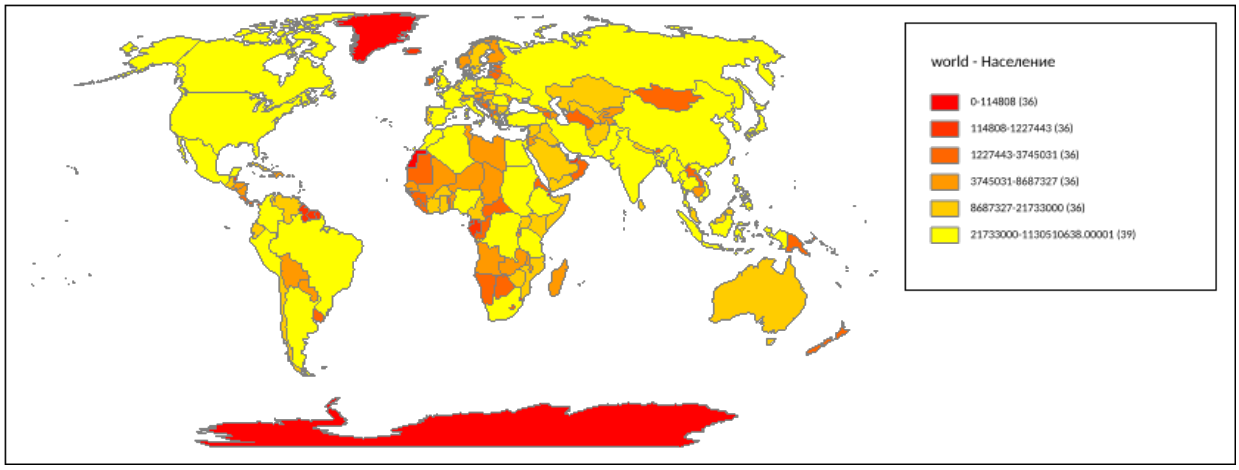
8.4 Легенда

Для вывода условных обозначений используется легенда. Легенда – таблица, содержащая образцы условных обозначений и письменных пояснений к ним. В Аксиоме.ГИС легенды карт представляются в отдельных окнах. Попробуем отрисовать в растре ранее созданные слой и тематику по интервалам. Заметим, что легенду также можно отрисовать на одном растре вместе с картой.

```
from axipy.render import Legend, Context
from PySide2.QtGui import QImage, QPainter

legend_world = Legend(lay_world)
legend_world.position = (10, 10)

legend_thematic = Legend(thematic)
legend_thematic.position = (200, 10)

image = QImage(500, 200, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter_legend = QPainter(image)
context_legend = Context(painter_legend)

legend_world.draw(context_legend)
legend_thematic.draw(context_legend)
painter_legend.end()
```

Создадим легенду, на этот раз сразу переведя в `PySide2.QtGui.QImage`, и скомпонуем с тематическим слоем, полученным ранее:

```
from axipy.render import Legend


legend_thematic = Legend(thematic)
legend_thematic.position = (10, 5)
legend_image = legend_thematic.to_image(200, 170)
```



8.5 Отчет

Для вывода информации на печать предусмотрено создание отчетов. Отчет формируется на базе стандартного подхода работы с принтером в Qt `PySide2.QtPrintSupport.QPrinter`. Создадим макет отчета, в который поместим геометрический объект и карту. Вывод сделаем в файл формата PDF. Для этого предварительно создадим объект принтера и установим необходимые свойства

```
from PySide2.QtPrintSupport import QPrinter

printer = QPrinter()
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName('../path/to/outdir/report.pdf')
```

Далее, создадим сам отчет и в конструктор передадим созданный ранее принтер.

```
from axipy.render import Report, GeometryReportItem, MapReportItem

report = Report(printer)
```

Создадим геометрический элемент и добавим его в отчет. Координаты в единицах измерения листа принтера.

```
geometryReportItem = GeometryReportItem()
geometryReportItem.geometry = Geometry.from_wkt(
        'POLYGON ((10 10, 10 100, 100 100, 50 50, 100 10, 10 10))')
geometryReportItem.style = Style.from_mapinfo(mapbasic.brush(45, 255, 65535))
report.items.add(geometryReportItem)
```

Аналогично добавим карту.

```
mapReportItem = MapReportItem(Rect(10, 120, 190, 220), map)
report.items.add(mapReportItem)
```

Контекст для печати по подобию рассмотренному контексту для карты.

```
from PySide2.QtGui import QPainter
from axipy.render import Context


painterReport = QPainter(printer)
context = Context(painterReport)
```

Производим печать.

```
report.draw(context)
painterReport.end()
```

В результате в файловой системе мы получим файл report.pdf, который содержит геометрический элемент и карту.

9 Интерфейс

9.1 Создание кнопок

Расположение кнопки в интерфейсе Аксиома.ГИС определяется Вкладной и Группой. Например, вкладка “Основные” группа “Команды”. В модуле `axipy.menubar` есть необходимые функции для создания кнопок.

```
from axipy import menubar


button = menubar.create_button('Простое действие', on_click=lambda: print('triggered
↪'))
position = menubar.get_position('Основные', 'Команды')
position.add(button)
```

Детально разберем, что делает этот пример.

Создается кнопка с текстом “Простое действие”, и ,используя параметр `on_click`, привязывается нажатие на кнопку к анонимной лямбде, которая печатает в консоль текст «*triggered*». Это обработчик нажатия кнопки. Обработчиком может служить любой callable-объект(функтор) без параметров, т.е. функции, лямбды, объекты с методом `__call__`. Также можно задать иконку кнопки параметром `icon=`. Иконкой может быть строка-ссылка на ресурс или объект типа `PySide2.QtGui.QIcon`.

Далее ищется расположение в интерфейсе. Если вкладка или группа с такими именами отсутствуют, то они будут созданы при добавлении кнопки.

В последней строке кнопка добавляется в заданное расположение.

10

Модули (Плагины)

Чтобы создать модуль, руководствуйтесь следующим:

1. Придумать идею - функционал, решающий какую-то проблему.
2. Создать структуру плагина - файлы и папки.
3. Написать код.
4. Протестировать.
5. Опубликовать.

Совет: Изучайте исходный код готовых модулей.

10.1 Структура модуля

Модуль для Аксиомы.ГИС это специально оформленный модуль Python с дополнительными файлами.

Рассмотрим структуру минимального модуля с обязательными параметрами и более расширенного:

Минимальный:

```
ru_mycompany_minimal_module  # папка с модулем
├── __init__.py  # точка входа
└── manifest.ini  # информация о модуле
```

Расширенный:

```
ru_mycompany_extended_module
├── documentation
│   └── index.html
├── business_logic.py
├── i18n
│   ├── translation_en.qm
```

(continues on next page)

(продолжение с предыдущей страницы)

```
│   └── translation_en.ts
├── __init__.py
├── manifest.ini
└── ui
    ├── form.ui
    ├── image.png
    └── logo.png
```

10.1.1 Идентификатор модуля

`ru_mycompany_minimal_module` - папка с модулем, она же - уникальный идентификатор модуля. Так же, как и при создании обычных модулей Python, избегайте конфликтов имен. Делайте имя модуля уникальным. Для этого следуйте простому соглашению именования: - используйте имя вашего веб-сайта или электронной почты, разделив на слова в обратном порядке; - используйте только маленькие латинские буквы и символ нижнего подчеркивания "_", т.е. `[a-z0-9_]`.

Так для модуля *mymodule* рекомендуемым идентификатором будет: - для веб-сайта *axioma-gis.ru* - `ru_axioma_gis_mymodule` - для почты andrey@yandex.ru - `ru_yandex_at_andrey_mymodule`

10.1.2 Точка входа

`__init__.py` - точка входа в модуль, так же как и для любого другого модуля Python - является обязательным для системы импорта. Содержит основной код модуля или импортирует другие локальные файлы.

См.также:

Точка входа может содержать специальный *Класс Plugin*.

10.1.3 Информация о модуле

`manifest.ini` - содержит основную информацию, версию, название и прочее. Является ini-файлом с простыми парами ключ=значение.

Примечание: Файл `manifest.ini` должен иметь кодировку UTF-8.

Минимальный пример содержимого:

```
[general]
name=Пример модуля
description=Короткий текст с описанием модуля.
```

Параметры:

- `name` - короткая строка с именем модуля
- `description` - короткий текст с описанием

См.также:

Для более подробного описания формата ini, поддерживаемого Аксиомой.ГИС, смотрите документацию configparser.

Например, могут содержаться другие необязательные параметры:

```
; может содержать коментарии
; следующая секция обязательна
[general]
name=Пример модуля
description=Короткий текст с описанием модуля.
    Может быть многострочным.
; конец обязательных параметров

; необязательные параметры
version=1.0
author=Андрей
email=andrey@axioma-gis.ru
homepage=https://andrey.axioma-gis.ru
repository=https://github.com/andey/mymodule
license=New BSD

; секция локализации
[i18n]
name_en=Plugin example
description_en=Plugin example description.
```

10.2 Документация

Документация может быть написана в HTML файлах. Аксиома.ГИС откроет документацию в системном веб-браузере. Аксиома ищет документацию в папке с модулем *documentation* - файлы `index[locale].html`. Пользователь откроет документацию с суффиксом локали, совпадающим с языком системы. При отсутствии совпадения будет открываться файл без суффикса - `index.html`.

10.3 Переводы

Можно предусмотреть загрузку модуля на разных языках.

Название и описание самого модуля может быть переведено на другие языки в манифесте в секции `i18n`:

```
; секция локализации
[i18n]
name_en=Plugin example
description_en=Plugin example description.
name_fr=Exemple de plugin
description_fr=Description de l'exemple de plugin.
```

У пользователя отобразится название и описание в случае, если язык системы будет совпадать с суффиксом локали. Иначе отобразится название и описание из основной секции `general`.

Наиболее простой способ создания и сопровождения переводов строк из исходного кода - использование «Qt Linguist».

Примечание: Подробнее о переводе в документации Qt Linguist.

Основные этапы:

1. Отмечаются строки, предназначенные для перевода.
2. Для экспорта строк используется утилита `lupdate`. Она проходит по файлам с исходным кодом и забирает все встречаемые строки, отмеченные для перевода. Результатом является файл с расширением `.ts` - простой структурированный xml файл со строками.
3. `.ts` - файл открывается в «Qt Linguist» и переводится на один или более языков.
4. После завершения перевода отдельных строк файл `.ts` “компилируется” в бинарный файл с расширением `.qm`, который будет загружен Аксиомой.ГИС. Для компиляции используется утилита `lrelease`.

```
lrelease your_plugin.ts
```

5. `.qm` - файлы размещаются в подпапке `i18n` внутри модуля. Они будут загружены вместе с модулем.

10.4 Класс Plugin

Модули рекомендуется писать в объектном стиле. Для этого точка входа `__init__.py` может содержать класс `Plugin`. Тогда Аксиома.ГИС при загрузке модуля создаст его экземпляр, а при выгрузке - уничтожит его.

Пример модуля `__init__.py`

```
"""Пример добавления кнопки и подключение действия по нажатию на нее
(показ сообщения). При выгрузке кнопка удаляется из интерфейса.
"""
from PySide2.QtWidgets import QMessageBox


class Plugin:
    def __init__(self, iface):
        self.iface = iface
        menubar = iface.menubar
        self.__action = menubar.create_button('Пример действия',
                icon=':/icons/share/32px/run.png', on_click=self.show_message)
        position = menubar.get_position('Основные', 'Команды')
        position.add(self.__action)

    def unload(self):
        self.iface.menubar.remove(self.__action)

    def show_message(self):
        QMessageBox.information(None, 'Сообщение',
                'Пример выполнения действия по нажатию кнопки')
```

- В конструктор `__init__` передается экземпляр объекта интерфейса Аксиомы *axipy.interface.AxiomaInterface* как параметр `iface`.
- `unload` - вызывается, когда модуль выгружается.

Вспомогательный класс *axipy.interface.AxiomaInterface* содержит свойства и функции, необходимые почти для любого плагина. Например, загрузка/сохранение настроек *settings*, получение файлов внутри папки с модулем *local_file()*, перевод строк *tr()*, добавление кнопок в интерфейс *menubar* и другое.

11

История изменений

11.1 3.0.0 Изменения

12 Апреля 2021

11.1.1 Новое

- Руководство разработчика объединено со справочником функций.
- Свойство временной таблицы *axipy.da.Table.is_temporary*.
- Менеджер контекста `with` для *axipy.da.DataObject*.
- Транзакционная модель редактирования таблиц: *axipy.da.Table.restore()*, *axipy.da.Table.commit()*, *axipy.da.Table.is_modified*, *axipy.da.Table.insert()*, *axipy.da.Table.update()*, `axipy.da.Table.delete()`.
- Каталог объектов данных *axipy.app.MainWindow.catalog* по-умолчанию. Открываемые объекты данных автоматически попадают в каталог главного окна. Запросы *axipy.da.DataCatalog.query()* производятся к этому каталогу без явного указания конкретных таблиц.
- Создаваемые окна `axipy.gui.ViewService.create_view()` автоматически добавляются в главное окно программы.
- Настройки Аксиомы *axipy.Settings*.
- Провайдеры данных *axipy.da.DataProviders* со специализированными параметрами для открытия/создания и импорта/экспорта: *tab*, *shp* и другие.
- Раздельные типы стилей: *axipy.da.PointStyle*, *axipy.da.PolygonStyle* и другие.
- Раздельные типы геометрий: *axipy.da.Point*, *axipy.da.Polygon* и другие.
- Загрузка/сохранение рабочих наборов *axipy.app.MainWindow.load_workspace()*, *axipy.app.MainWindow.save_workspace()*.

11.1.2 Исправления

- Ошибка при попытке закрытия временной таблицы с изменениями.
- Ошибка при задании разделителя в формате CSV *axipy.da.CsvDataProvider*.

11.2 2.9.0 Изменения

15 Декабря 2020

11.2.1 Новое

- Первоначальный релиз.

12

Справочник функций

Справочник описывает модули и функции основного модуля **axipy** - нового API для Аксиомы.ГИС на языке Python.

Примечание: Не путать с модулем **axioma**; документация к нему расположена в другом месте.

12.1 Модули Аксиома.ГИС

12.1.1 axipy

Основной пакет API для взаимодействия с Аксиомой.ГИС.

Предоставляет доступ к Аксиоме.ГИС через набор модулей, подмодулей, классов и функций.

`axipy.`**`io`**
Объект открытия/создания объектов данных.

Type *axipy.da.DataProviders*

Функции

`axipy.`**`init_axioma`**`()`
Инициализирует ядро Аксиомы.ГИС.

Тип результата `QApplication`

Результат Приложение Qt5 с очередью событий (event-loop).

Пример:

```
app = init_axioma()
app.exec_()  # запускает обработку очереди событий
```

class axipy.**Settings**

Настройки Аксиомы.

Список 1: Пример использования

```
# Читает значение
val = Settings.value(Settings.RulerColorLine)
# Записывает значение
new_value = QColor(0, 255, 0)
Settings.setValue(Settings.RulerColorLine, new_value)
# Сбрасывает на значение по-умолчанию
Settings.reset(Settings.RulerColorLine)
```

Таблица 1: Атрибуты

Значение	Тип	Наименование
SilentCloseWidget	*bool*	Подтверждать закрытие несохраненных данных
SnapSensitiveRasius	int	Привязка узлов - размер
SnapColor	QColor	Привязка узлов - цвет
SnapThickness	int	Привязка узлов - толщина линии
EditNodeColor	QColor	Узлы при редактировании - цвет
EditNodeSize	int	Узлы при редактировании - размер
NearlyGeometriesTopology	*bool*	Перемещать узлы соседних объектов при редактировании
NodesUpdateMode	*bool*	Использовать перезапись истории изменений при редакти
ShowDrawingToolTip	*bool*	Показывать данные при рисовании
CreateTabAfterOpen	*bool*	Создавать TAB при открытии
RenameDataObjectFromTab	*bool*	Переименовывать открытый объект по имени TAB файла
LastSavePath	str	Последний пусть сохранения
UseLastSelectedFilter	*bool*	Запоминать последний фильтр в диалоге открытия файло
SelectByInformationTool	*bool*	Инструмент «Информация» выбирает объект
SaveAsToOriginalFileFolder	*bool*	Сохранять копию в каталог с исходным файлом
LastNameFilter	str	Последний использованный фильтр файлов
SensitiveMouse	*bool*	Чувствительность мыши
ShowSplashScreen	*bool*	Отображать экран загрузки
ShowSplashScreenMessages	*bool*	Отображать сообщения экрана загрузки
SplashScreenImageFile	str	Файл изображения
RulerModeSpherical	*bool*	Линейка - измерение на сфере
RulerColorLine	*bool*	Линейка - цвет линии
UseAntialiasing	*bool*	Использовать сглаживание при отрисовке
ShowDegreeTypeNumeric	*bool*	Отображать градусы в формате Десятичное значение
DrawCoordSysBounds	*bool*	Отображать границы мира
PreserveScaleMap	*bool*	Сохранять масштаб при изменении размеров окна
ShowMapScaleBar	*bool*	Показывать масштабную линейку
ShowScrollOnMapView	*bool*	Показывать полосы прокрутки
LoadLastWorkspace	*bool*	Загружать при старте последнее рабочее пространство
ShowMeshLayout	*bool*	Отображать сетку привязки
MeshSizeLayout	*float*	Размер ячейки
SnapToMeshLayout	*bool*	Привязывать элементы отчета к сетке
ShowMeshLegend	*bool*	Отображать сетку привязки
MeshSizeLegend	*float*	Размер ячейки
SnapToMeshLegend	*bool*	Привязывать к сетке
LastOpenPath	str	Последний каталог откуда открывались данные
LastPathWorkspace	str	Последний каталог к рабочему набору

continu

Таблица 1 – продолжение с предыдущей страницы

Значение	Тип	Наименование
DefaultPathCache	str	Каталог с кэшированными данными
UserDataPaths	list[str]	Список пользовательских каталогов с названиями
EnableSmartTabs	bool	Умное переключение вкладок
DistancePrecision	int	Точность по умолчанию для расстояний и площадей

classmethod reset(*key*)
Устанавливает значение по умолчанию.

Параметры **key** – Параметр.

classmethod setValue(*key, value*)
Устанавливает значение параметра.

Параметры

- **key** – Параметр.
- **value** – Значение.

Исключение TypeError – Если значение неправильного типа.

Например:

```
Settings.setValue(Settings.LastOpenPath, 'C:/mydir')
```

classmethod value(*key*)
Читает значение параметра.

Параметры **key** – Параметр.

Тип результата Any

Результат Значение.

Например:

```
val = Settings.value(Settings.LastOpenPath)
```

class axipy.interface.**AxiomaInterface**(*plugin_dir*)
Интерфейс для модуля.

Вспомогательный класс для создания модулей.

См.также:

Подробнее в главе *Модули (Плагины)*.

property catalog
Хранилище объектов данных.

Тип результата *DataCatalog*

property io
Класс открытия/создания объектов данных.

Тип результата *DataProviders*

local_file(**paths*)
Возвращает путь к файлу/папке относительно модуля.

Параметры ***path** – Составные относительного пути.

Тип результата str

Результат Абсолютный путь.

Пример:

```
plugin_path = iface.local_file()
icon_path = iface.local_file('images', '32px', 'logo.png')
```

property menubar
Объект с функциями меню главного окна Аксиомы.ГИС.

См.также:

axipy.menubar

property settings
Настройки модуля.

Позволяет сохранять и загружать параметры.

См.также:

Подробнее в документации на класс PySide2.QtCore.QSettings.

Тип результата QSettings

tr(*text*)
Ищет перевод строки строки.

Производит поиск строки в загруженных файлах перевода.

Параметры **text** (str) – Строка для перевода.

Тип результата str

Результат Перевод стоки, если строка найдена. Иначе - сама переданная строка.

Пример:

```
button_name = iface.tr('My button')
```

12.1.2 axipy.app

Модуль приложения.

Данный модуль является основным модулем приложения.

axipy.app.**mainwindow**
Готовый экземпляр главного окна Аксиомы.

Type *MainWindow*

12.1.2.1 Главное окно приложения - MainWindow

class axipy.app.**MainWindow**
Главное окно Аксиомы.

Примечание: Используйте готовый объект *axipy.app.mainwindow*.

---

add(*view*)
Добавляет окно просмотра данных.

Параметры **view** (*View*) – окно просмотра данных.

Примечание: При создании окон просмотра данных *axipy.gui.ViewService.create_mapview()* или *axipy.gui.ViewService.create_tableview()* они автоматически добавляются в главное окно программы.

Тип результата QMdiSubWindow

property catalog
Хранилище объектов приложения.

Это то же хранилище, которое отображается в панели «Открытые данные».

Примечание: При открытии объектов данных *axipy.da.DataProviders.openfile()* они автоматически попадают в каталог.

Тип результата *DataCatalog*

load_workspace(*fileName*)
Читает рабочее пространство из файла.

Параметры **fileName** (str) – Наименование входного файла.

qt_object()
Возвращает Qt5 объект окна.

Тип результата QMainWindow

save_workspace(*fileName*)
Сохраняет рабочее пространство в файл.

Параметры **fileName** (str) – Наименование выходного файла.

12.1.3 axipy.cs

Модуль систем координат.

В данном модуле содержатся классы и методы, предназначенные для удобной работы с координатными системами.

axipy.cs.**unit**
Экземпляр класса, позволяющий получить доступ к единицам измерения.

Type *UnitService*

См.также:

Подробнее об использовании см. *LinearUnit* и *AreaUnit*

12.1.3.1 Система Координат (СК) - CoordSystem

class axipy.cs.**CoordSystem**
Система координат (СК). СК описывает каким образом реальные объекты на земной поверхности могут быть представлены в виде двумерной проекции. Выбор СК для представления данных зависит от конкретных исходных условий по представлению исходных данных.

Поддерживается создание СК посредством следующих вариантов:

- Из строки MapInfo PRJ *from_prj()*
- Из строки PROJ *from_proj()*
- Из строки WKT *from_wkt()*
- Из значения EPSG *from_epsg()*
- План/Схему с указанием единиц измерения и охвата *from_units()*

Примеры:

```
cs_epsg = CoordSystem.from_epsg(4326)
cs_prj = CoordSystem.from_prj('1, 104')
cs_proj = CoordSystem.from_proj('+proj=longlat +ellps=WGS84 +no_defs')
cs_wkt = CoordSystem.from_wkt('GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID[
↪"WGS_1984",6378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["Degree",0.
↪0174532925199433295]]')
```

convert_from_degree(*value*)
Переводит из градусов в единицы измерения системы координат.

Тип результата Union[*Pnt*, List[*Pnt*], *Rect*]

convert_to_degree(*value*)
Переводит из единиц измерения системы координат в градусы.

Пример:

```
csMercator = CoordSystem. from_prj("10, 104, 7, 0")
p_out = csMercator.to_degree((1000000, 1000000))
print(p_out)

>>> (8.983152841195214 9.005882635078796)
```

Тип результата Union[*Pnt*, List[*Pnt*], *Rect*]

property description
Краткое текстовое описание.

Тип результата str

property epsg
Значение EPSG если существует для данной системы координат, иначе None.

Тип результата Optional[int]

classmethod from_epsg(*code*)
Создает координатную систему по коду EPSG.

См.также:

Подробнее см. EPSG

Параметры code (int) – Стандартное значение EPSG.

Тип результата *CoordSystem*

classmethod from_prj(*prj*)
Создает координатную систему из строки MapBasic.

См.также:

Подробнее см. PRJ

Параметры prj (str) – Строка MapBasic. Допустима короткая нотация.

Пример:

```
csMercator = CoordSystem.from_prj('10, 104, 7, 0')
csLatLon = CoordSystem.from_prj('Earth Projection 1, 104')
csMercator = CoordSystem.from_prj('NonEarth 0, 'm'')
```

Тип результата *CoordSystem*

classmethod from_proj(*proj*)
Создает координатную систему из строки proj.

См.также:

Подробнее см. PROJ

Параметры proj (str) – Строка proj.

Тип результата *CoordSystem*

classmethod from_string(*string*)
Создает систему координат из строки.

Строка состоит из двух частей: префикса и строки представления СК. Возможные значения префиксов: «proj», «wkt», «epsg», «prj».»

Параметры string (str) – Строка.

Пример:

```
crs1 = CoordSystem.from_string('epsg:4326')
crs2 = CoordSystem.from_string('prj:1,104')
```

Тип результата *CoordSystem*

classmethod from_units(*unit*, *rect=<axipy.utl.Rect object>*)
Создает декартову систему координат.

Параметры

- **unit** (*LinearUnit*) – Единицы измерения системы координат.
- **rect** (Union[*Rect*, QRectF, None]) – Охват системы координат.

Пример:

```
ne = CoordSystem.from_units(unit.km, Rect(-100, -100, 100, 100))
```

Тип результата *CoordSystem*

classmethod from_wkt(*wkt*)
Создает координатную систему из строки WKT.

См.также:

Подробнее см. WKT

Параметры **wkt** (str) – Строка WKT.

Тип результата *CoordSystem*

property lat_lon
Является ли данная СК широтой/долготой.

Тип результата bool

property name
Наименование системы координат.

Тип результата str

property non_earth
Является ли данная СК декартовой.

Тип результата bool

property prj
Строка prj формата MapBasic или пустая строка, если аналога не найдено.

Тип результата str

property proj
Строка PROJ или пустая строка, если аналога не найдено.

Тип результата str

property rect
Максимально допустимый охват.

Тип результата *Rect*

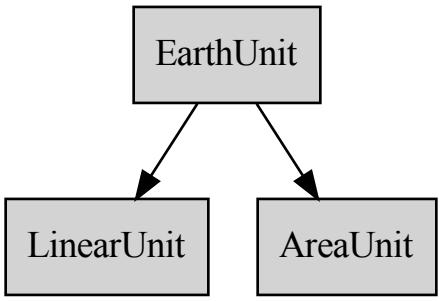
property unit
Единицы измерения.

Тип результата *LinearUnit*

property wkt
Строка WKT или пустая строка, если аналога не найдено.

Тип результата str

12.1.3.2 Единицы измерения - EarthUnit



class axipy.cs.**EarthUnit**
Базовый абстрактный класс единиц измерения.

property conversion
Коэффициент преобразования в метры.

Тип результата float

property description
Краткое описание.

Тип результата str

property localized_name
Локализованное краткое наименование единиц измерения.

Тип результата str

property name
Краткое наименование единиц измерения.

Тип результата str

to_unit(*unit*, *value=1*)
Перевод значения в другие единицы измерения.

Параметры

- **unit** (Union[*LinearUnit*, *AreaUnit*]) – Единицы измерения, в которые необходимо перевести значение.
- **value** (float) – Значение для перевода.

Пример:

```
from axipy import *

print("Linear:", unit.km.to_unit(unit.m, 2))
print("Area:", unit.sq_km.to_unit(unit.sq_m, 2))

>>> Linear: 2000.0
>>> Area: 2000000.0
```

Тип результата float

Единицы измерения расстояний - LinearUnit

class axipy.cs.**LinearUnit**

Базовые классы: *axipy.cs.EarthUnit*

Линейные единицы измерения. Используются для работы с координатами объектов или расстояний.

Примечание: Получить экземпляр можно через сервис *axipy.cs.unit* по соответствующему атрибуту.

Перечень в виде списка можно получить посредством axipy.cs.unit.all_linear()

km
Километры

Type *LinearUnit*

m
Метры

Type *LinearUnit*

mm
Миллиметры

Type *LinearUnit*

cm
Сантиметры

Type *LinearUnit*

mi
Мили

Type *LinearUnit*

nmi
Морские мили.

Type *LinearUnit*

inch
Дюймы

Type *LinearUnit*

ft
Футы

Type *LinearUnit*

yd
Ярды

Type *LinearUnit*

survey_ft
Топографические футы.

Type *LinearUnit*

li
Линки

Type *LinearUnit*

ch
Чейны

Type *LinearUnit*

rd
Роды

Type *LinearUnit*

Примеры:

```
print(axipy.unit.km.description)

>>> километры

for u in unit.all_linear:
    print('unit name: {}'.format(u.description))

>>> unit name: километры
>>> unit name: метры
>>> unit name: миллиметры
>>> unit name: сантиметры
>>> unit name: мили
>>> unit name: морские мили
>>> unit name: дюймы
>>> unit name: футы
>>> unit name: ярды
>>> unit name: топографические футы
>>> unit name: линки
>>> unit name: чейны
>>> unit name: роды
```

Единицы измерения площадей - AreaUnit

class axipy.cs.**AreaUnit**

Базовые классы: *axipy.cs.EarthUnit*

Единицы измерения площадей.

Примечание: Получить экземпляр можно через сервис *axipy.cs.unit* по соответствующему атрибуту.

Перечень в виде списка можно получить посредством `axipy.cs.unit.all_area()`

sq_km
Квадратные километры

Type *AreaUnit*

sq_m
Квадратные метры

Type *AreaUnit*

sq_mm
Квадратные миллиметры

Type *AreaUnit*

sq_cm
Квадратные сантиметры

Type *AreaUnit*

sq_mi
Квадратные мили

Type *AreaUnit*

sq_nmi
Квадратные морские мили.

Type *AreaUnit*

sq_inch
Квадратные дюймы

Type *AreaUnit*

sq_ft
Квадратные футы

Type *AreaUnit*

sq_yd
Квадратные ярды

Type *AreaUnit*

sq_survey_ft
Квадратные топографические футы.

Type *AreaUnit*

sq_li
Квадратные линки

Type *AreaUnit*

sq_ch
Квадратные чейны

Type *AreaUnit*

sq_rd
Квадратные роды

Type *AreaUnit*

Пример:

```
print(axipy.unit.sq_km.description)

>>> кв. километры
```

Сервис доступа к единицам измерения - UnitService

class axipy.cs.**UnitService**

property all_area
Перечень всех площадных единиц измерения.

Тип результата List[*AreaUnit*]

property all_linear
Перечень всех линейных единиц измерения.

Тип результата List[*LinearUnit*]

12.1.3.3 Трансформация координат - CoordTransformer

class axipy.cs.**CoordTransformer**(*cs_from*, *cs_to*)
Класс для преобразования координат из одной СК в другую. При создании объекта трансформации в него передается исходная и целевая СК. После этого данный объект может использоваться для преобразования данных между этими СК.

Параметры

- **cs_from** (Union[*CoordSystem*, str]) – Исходная СК.
- **cs_to** (Union[*CoordSystem*, str]) – Целевая СК.

Пример:

```
# Пример преобразования точки
from axipy import *

csLL = CoordSystem.from_prj("1, 104")
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
inPoint = Pnt(10, 10)
transformer = CoordTransformer(csLL, csMercator)
outPoint = transformer.transform(inPoint)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print('Result point:', outPoint)
outRect = transformer.transform(Rect(0,0,10,10))
print('Result rect:', outRect)
```

transform(*value*)

Преобразовывает точки из исходной СК в целевую СК.

Параметры value (Union[*Pnt*, List[*Pnt*], QPointF, QRectF, *Rect*, List[QPointF]]) – Входное значение. Может быть точкой, массивом точек *axipy.utl.Pnt* или *axipy.utl.Rect*.

Тип результата Union[*Pnt*, *Rect*, List[*Pnt*]]

Результат Выходное значение. Тип зависит от входного и аналогичен ему.

Исключение RuntimeError – Ошибка выполнения преобразования.

12.1.4 axipy.da

Модуль источников данных.

В данном модуле содержатся классы и методы для работы с источниками данных.

12.1.4.1 Объект открытия/создания данных - DataProviders

class axipy.da.**DataProviders**

Класс открытия/создания объектов данных.

Примечание: Используйте готовый экземпляр этого класса *axipy.io*

Примечание: Для удобного задания параметров используйте экземпляры провайдеров: *tab*, *shp*, *csv*, *mif*, *excel*, *sqlite*, *postgre*, *oracle*.

Примечание: Открытые данные автоматически попадают в хранилище данных *axipy.da.DataCatalog*.

Пример:

```
table = axipy.io.openfile('../path/to/datadir/table.tab')
```

create(*definition*)

Создает и открывает данные из описания.

Параметры definition (dict) – Описание объекта данных.

Псевдоним *create_open()*.

Тип результата *DataObject*

create_open(*definition*)
Создает и открывает данные из описания.

Возможные параметры:

- `src` - Строка, определяющая местоположение источника данных. Это может быть либо путь к файлу с расширением TAB, либо пустая строка (для таблицы, размещаемой в памяти).
- `schema` - Схема таблицы. Задается массивом объектов, содержащих атрибуты.

Параметры **definition** (`dict`) – Описание объекта данных.

Пример:

```
definition = {
    'src': '../path/to/datadir/edit/table.tab',
    'schema': attr.schema(
        attr.string('field1'),
        attr.integer('field2'),
    ),
}
table = io.create(definition)
```

Тип результата *DataObject*

createfile(*filepath*, *schema*, **args*, ***kwargs*)
Создает таблицу.

create() выполняет ту же функцию, но в более обобщенном виде.

Параметры

- **filepath** (`str`) – Путь к создаваемой таблице.
- **schema** – Схема таблицы.

Тип результата *DataObject*

property csv
Файловый провайдер - Текст с разделителями.

Тип результата *CsvDataProvider*

property excel
Провайдер чтения файлов Excel.

Тип результата *ExcelDataProvider*

loaded_providers()
Возвращает список всех загруженных провайдеров данных.

Тип результата `dict`

Результат Провайдеры в виде пар (`Идентификатор : Описание`).

property mif
Провайдер данных MIF-MID.

Тип результата *MifMidDataProvider*

open(*definition*)
Открывает данные по описанию.

Формат описания объектов данных индивидуален для каждого провайдера данных, однако многие элементы используются для всех провайдеров данных.

Параметры **definition** (dict) – Описание объекта данных.

Пример:

```
# Пример открытия GPKG файла::
definition = { 'src': '../path/to/datadir/example.gpkg',
               'dataobject': 'tablename',
               'provider': 'SqliteDataProvider' }
table = io.open(definition)
```

Пример открытия таблицы базы данных:

```
definition = {"src": "localhost",
              "db": "sample",
              "user": "postgres",
              "password": "postgres",
              "dataobject": "public.world",
              "provider": "PgDataProvider"}
table = io.open(definition)
```

Тип результата *DataObject*

openfile(*filepath*, **args*, ***kwargs*)
Открывает данные из файла.

Параметры

- **filepath** (str) – Путь к открываемому файлу.
- ****kwargs** – Именованные аргументы.

Пример:

```
table = io.openfile('../path/to/datadir/example.gpkg')
```

Тип результата *DataObject*

property oracle
PostgreSQL.

Тип результата *OracleDataProvider*

property postgre
PostgreSQL.

Тип результата *PostgreDataProvider*

query(*query_text*, **tables*)
Выполняет SQL-запрос к перечисленным таблицам.

Предупреждение: Используйте *axipy.da.DataCatalog.query()*.

Параметры

- **query_text** (str) – Текст запроса.
- ***tables** – Список таблиц, к которым выполняется запрос.

Тип результата *Table*

Результат Таблица, если результатом запроса является таблица.

Пример:

```
query_text = "SELECT * FROM world, caps WHERE world.capital = caps.capital"
joined = io.query(query_text, world, caps)
```

read_contents(*definition*)

Читает содержимое источника данных.

Обычно используется для источников, способных содержать несколько объектов данных.

Параметры definition (Union[dict, str]) – Описание источника данных.

Тип результата List[str]

Результат Имена объектов данных.

Пример:

```
>>> io.read_contents('../path/to/datadir/example.gpkg')
['world', 'worldcap']

>>> world = io.openfile('../path/to/datadir/example.gpkg',
...          dataobject='world')
```

property shp

Векторный провайдер OGR.

Тип результата *ShapeDataProvider*

property sqlite

Векторный провайдер sqlite.

Тип результата *SqliteDataProvider*

property tab

Провайдер MapInfo.

Тип результата *TabDataProvider*

class axipy.da.**Source**(**args*)

Источник данных.

Используется для открытия данных или для указания источника при конвертации.

Пример открытия:

```
table = source.open()
```

Пример конвертации:

```
destination.export_from(source)
```

Примечание: Не все провайдеры поддерживают открытие и конвертацию. См. описание конкретного провайдера данных.

---

open()
Открывает источник данных.

Тип результата *DataObject*

class axipy.da.**Destination**(*schema*, **args*)
Назначение объекта данных.

Используется для создания данных или для указания назначения при конвертации.

Пример создания:

```
table = destination.create_open()
```

Пример конвертации:

```
destination.export_from(source)
```

Примечание: Не все провайдеры поддерживают создание и конвертацию. См. описание конкретного провайдера данных.

---

create_open()
Создает и открывает объект данных.

Тип результата *DataObject*

export(*features*)
Создает объект данных и экспортирует в него записи.

Параметры **features** (Iterator[*Feature*]) – Записи.

export_from(*source*, *copy_schema=False*)
Создает объект данных и экспортирует в него записи из источника данных.

Параметры

- **source** (*Source*) – Источник данных.
- **copy_schema** (bool) – Копировать схему источника без изменений.

export_from_table(*table*, *copy_schema=False*)
Создает объект данных и экспортирует в него записи из таблицы.

Параметры

- **table** (*Table*) – Таблица.
- **copy_schema** (bool) – Копировать схему источника без изменений.

class axipy.da.**DataProvider**(*info*)
Абстрактный провайдер данных.

create(**args*, ***kwargs*)
Создает и открывает источник данных.

Псевдоним *create_open()*.

create_open(**args*, ***kwargs*)
Создает и открывает источник данных.

Пример:

```
provider.create_open(...)
```

Что эквивалентно:

```
provider.get_destiantion(...).create_open()
```

См.также:

DataProvider.destination().

file_extensions()
Список поддерживаемых расширений файлов.

Тип результата List[str]

Результат Пустой список для не файловых провайдеров.

get_destination()
Создает назначение объекта данных.

Исключение NoteImplementedError – Если провайдер не поддерживает создание назначений.

Тип результата *Destination*

get_source()
Создает источник данных.

Исключение NoteImplementedError – Если провайдер не поддерживает создание источников.

Тип результата *Source*

property id
Идентификатор провайдера.

Тип результата str

open(**args*, ***kwargs*)
Открывает источник данных.

Пример:

```
provider.open(...)
```

Что эквивалентно:

```
provider.get_source(...).open()
```

См.также:

DataProvider.source().

class axipy.da.**CsvDataProvider**(*info*)
Файловый провайдер: Текст с разделителями.

get_destination(*filepath*, *schema*, *with_header=True*, *delimiter=','*, *encoding='utf8'*)
Создает назначение объекта данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **schema** (*Schema*) – Схема таблицы.
- **with_header** (bool) – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** (str) – Разделитель полей.
- **encoding** (str) – Кодировка.

Тип результата *Destination*

get_source(*filepath*, *with_header=True*, *delimiter=','*, *encoding='utf8'*)
Создает источник данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **with_header** (bool) – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** (str) – Разделитель полей.
- **encoding** (str) – Кодировка.

Тип результата *Source*

class axipy.da.**ExcelDataProvider**(*info*)
Провайдер чтения файлов Excel.

get_destination()

> **Внимание:** Не поддерживается.

Тип результата *Destination*

get_source(*filepath*, *page*, *with_header=False*, *encoding='utf8'*)
Создает источник данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **page** (str) – Имя страницы.
- **with_header** (bool) – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **encoding** (str) – Кодировка.

Тип результата *Source*

class axipy.da.**MifMidDataProvider**(*info*)
Провайдер данных MIF-MID.

convert_to_tab(*mif_filepath*, *tab_filepath*)
Конвертирует из MIF в TAB.

Параметры

- **mif_filepath** (str) – Путь к исходному файлу.
- **tab_filepath** (str) – Путь к выходному файлу.

get_destination(*filepath*, *schema*)
Создает назначение объекта данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **schema** (*Schema*) – Схема таблицы.

Тип результата *Destination*

get_source()

> **Внимание:** Не поддерживается.
>
> Поддерживает экспорт только в TAB. См. *convert_to_tab()*.

Тип результата *Source*

class axipy.da.**ShapeDataProvider**(*info*)

get_destination(*filepath*, *schema*, *encoding='utf8'*)
Создает назначение объекта данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **schema** (*Schema*) – Схема таблицы.
- **encoding** (str) – Кодировка.

Тип результата *Destination*

get_source(*filepath*, *encoding='utf8'*)
Создает источник данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **encoding** (str) – Кодировка.

Тип результата *Source*

open_temporary(*schema*)
Создает временную таблицу.

Параметры **schema** (*Schema*) – Схема таблицы.

Тип результата *Table*

class axipy.da.**SqliteDataProvider**(*info*)
Векторный провайдер sqlite.

get_destination()

> **Внимание:** Не поддерживается.

Тип результата *Destination*

get_source(*filepath*, *dataobject*)
Создает источник данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **dataobject** (str) – Имя таблицы.

Тип результата *Source*

class axipy.da.**TabDataProvider**(*info*)
Провайдер MapInfo.

get_destination(*filepath*, *schema*)
Создает назначение объекта данных.

Параметры

- **filepath** (str) – Путь к файлу.
- **schema** (*Schema*) – Схема таблицы.

Тип результата *Destination*

get_source(*filepath*)
Создает источник данных.

Параметры **filepath** (str) – Путь к файлу.

Тип результата *Source*

class axipy.da.**PostgreDataProvider**(*info*)
PostgreSQL.

get_destination(*schema*, *dataobject*, *db_name*, *host*, *user*, *password*, *port=5432*)
Создает назначение объекта данных.

Параметры

- **schema** (*Schema*) – Схема таблицы.
- **dataobject** (str) – Имя таблицы.
- **db_name** (str) – Имя базы данных.
- **host** (str) – Адрес сервера.
- **user** (str) – Имя пользователя.
- **password** (str) – Пароль.
- **port** (int) – Порт.

Тип результата *Destination*

get_source(*db_name*, *host*, *user*, *password*, *port=5432*)
Создает источник данных.

Параметры

- **dataobject** (str) – Имя таблицы.

- **db_name** (str) – Имя базы данных.
- **host** (str) – Адрес сервера.
- **user** (str) – Имя пользователя.
- **password** (str) – Пароль.
- **port** (int) – Порт.

Тип результата *Source*

class axipy.da.**OracleDataProvider**(*info*)
Oracle.

> **Внимание:** Для подключения к БД Oracle необходимо настроить `Oracle Instant Client`. См. Руководство по установке и активации.

get_destination(*schema*, *dataobject*, *db_name*, *host*, *user*, *password*, *port=5432*)
Создает назначение объекта данных.

Параметры

- **schema** (*Schema*) – Схема таблицы.
- **dataobject** (str) – Имя таблицы.
- **db_name** (str) – Имя базы данных.
- **host** (str) – Адрес сервера.
- **user** (str) – Имя пользователя.
- **password** (str) – Пароль.
- **port** (int) – Порт.

Тип результата *Destination*

get_source(*db_name*, *host*, *user*, *password*, *port*)
Создает источник данных.

Параметры

- **dataobject** (str) – Имя таблицы.
- **db_name** (str) – Имя базы данных.
- **host** (str) – Адрес сервера.
- **user** (str) – Имя пользователя.
- **password** (str) – Пароль.
- **port** – Порт.

Тип результата *Source*

12.1.4.2 Каталог данных - DataCatalog

class axipy.da.**DataCatalog**
Хранилище объектов данных.

Примечание: Используйте готовый экземпляр этого класса *axipy.app.MainWindow.catalog*.

---

add(*data_object*)
Добавляет объект данных в хранилище.

Параметры **data_object** (*DataObject*) – Объект данных для добавления.

property added
Signal[str] Сигнал о добавлении объекта.

Тип результата Signal

count()
Возвращает количество объектов данных.

Тип результата int

find(*name*)
Производит поиск объект данных по имени.

Параметры **name** (str) – Имя объекта данных.

Тип результата Optional[*DataObject*]

Результат Искомый объект данных или None.

objects()
Возвращает список объектов.

Тип результата List[*DataObject*]

query(*query_text*)
Выполняет SQL-запрос к перечисленным таблицам.

Параметры **query_text** (str) – Текст запроса.

Тип результата Optional[*Table*]

Результат Таблица, если результатом запроса является таблица.

Пример:

```
query_text = "SELECT * FROM world, caps WHERE world.capital = caps.capital"
joined = catalog.query(query_text)
```

remove(*data_object*)
Удаляет объект данных.

Объект данных при этом закрывается.

Параметры **data_object** (*DataObject*) – Объект данных для удаления.

remove_all()
Удаляет все объекты данных.

property removed
Signal[str] Сигнал об удалении объекта.

Тип результата Signal

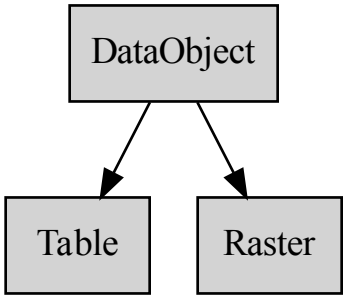
tables()
Возвращает список таблиц.

Тип результата List[*Table*]

property updated
Signal[] Сигнал об изменении количества объектов.

Тип результата Signal

12.1.4.3 Объект данных - DataObject



class axipy.da.**DataObject**
Объект данных.

Открываемые объекты из источников данных представляются объектами этого типа. Возможные реализации: таблица, растр, грид, чертеж, панорама, и так далее.

Пример:

```
table = io.openfile('path/to/file.tab')
...
table.close()  # Закрывает таблицу
```

Для закрытия объекта данных можно использовать менеджер контекста - выражение with. См. *close()*.

Пример:

```
with io.openfile('path/to/file.tab') as raster:
    ...
    # При выходе из блока растр будет закрыт
```

close()
Пытается закрыть таблицу.

Исключение RuntimeError – Ошибка закрытия таблицы.

Примечание: Объект данных не всегда может быть сразу закрыт. Например, для таблиц используется транзакционная модель редактирования и перед закрытием необходимо сохранить или отменить изменения, если они есть. См. *Table.is_modified*.

property name
Название объекта данных.

Тип результата str

property provider
Провайдер изначального источника данных.

Тип результата str

Таблица - Table

class axipy.da.**Table**

Базовые классы: *axipy.da.DataObject*

Таблица.

Менеджер контекста сохраняет изменения и закрывает таблицу.

Пример:

```
with io.openfile('path/to/file.tab') as table:
    ...
    # При выходе из блока таблица будет сохранена и закрыта
```

См.также:

commit(), *DataObject.close()*.

commit()
Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение RuntimeError – Невозможно сохранить изменения.

property coordsystem
Система координат таблицы.

Тип результата Optional[*CoordSystem*]

count(*bbox=None*)
Возвращает количество записей, удовлетворяющих параметрам.

Данный метод является наиболее предпочтительным для оценки количества записей. При этом используется наиболее оптимальный вариант выполнения запроса для каждого конкретного провайдера данных.

Параметры bbox (Union[*Rect*, QRectF, tuple, None]) – Ограничивающий прямоугольник.

Тип результата int

Результат Количество записей.

insert(*features*)
Вставляет записи в таблицу.

Параметры **features** (Union[Iterator[*Feature*], *Feature*]) – Записи для вставки.

property is_editable
Признак того, что таблица является редактируемой.

Тип результата bool

property is_modified
Таблица содержит несохраненные изменения.

Тип результата bool

property is_spatial
Признак того, что таблица является пространственной.

Тип результата bool

property is_temporary
Признак того, что таблица является временной.

Тип результата bool

items(*bbox=None*, *ids=None*)
Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** (Union[*Rect*, QRectF, tuple, None]) – Ограничивающий прямоугольник.
- **ids** (Optional[List[int]]) – Список идентификаторов.

Тип результата Iterator[*Feature*]

Результат Итератор по записям.

itemsByIds(*ids*)
Запрашивает записи по списку list с идентификаторами записей, либо перечень идентификаторов в виде списка.

Параметры **ids** (List[int]) – Список идентификаторов.

Тип результата Iterator[*Feature*]

Результат Итератор по записям.

Пример запроса по прямоугольнику (таблица в проекции *Робинсона*):

```
table_world = io.openfile('world.tab')
items = table_world.itemsByIds([11,27,41,163,203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(*obj*)
Запрашивает записи с фильтром по геометрическому объекту.

Параметры **obj** (*Geometry*) – Геометрия. Если для нее не задана СК, используется СК таблицы.

Тип результата Iterator[*Feature*]

Результат Итератор по записям.

Пример запроса по полигону:

```
table_world = io.openfile('world.tab')
v = 2000000
polygon = Polygon((-v, -v), (-v, v), (v, v), (v, -v))
items = table_world.itemsInObject(polygon)
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInRect(*bbox*)

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры bbox (Union[*Rect*, QRectF, tuple]) – Ограничивающий прямоугольник.

Тип результата Iterator[*Feature*]

Результат Итератор по записям.

Пример запроса (таблица в проекции *Робинсона*):

```
table_world = io.openfile('world.tab')
v = 2000000
items = table_world.itemsInRect(Rect(-v, -v, v, v))
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

remove(*ids*)

Удаляет записи из таблицы.

Параметры ids (Union[int, Iterator[int]]) – Идентификаторы записей для удаления.

restore()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema

Возвращает схему таблицы.

Тип результата *Schema*

update(*features*)

Обновляет записи в таблице.

Параметры features (Union[Iterator[*Feature*], *Feature*]) – Записи для обновления.

При обновлении проверяется *Feature.id*. Если запись с таким идентификатором не найдена, то она пропускается.

Список 2: Пример использования

```
modified_feature = Feature({'attr_name': 'new_value'}, id=1)
table.update(modified_feature)
table.commit()
```

См.также:

Feature.id, *commit()*, *is_modified*.

Атрибут записи - Attribute

class axipy.da.**Attribute**(*name*, *typedef*)

Атрибут схемы таблицы.

Используется для создания и инспектирования атрибутов и схем.

Параметры

- **name** (str) – Название.
- **typedef** (str) – Описание типа.

Совет: Для создания атрибутов используйте функции класса *axipy.da.AttributeFactory*: *axipy.da.AttributeFactory.string()*, *axipy.da.AttributeFactory.decimal()* и другие.

См.также:

axipy.da.Schema.

property length

Длина атрибута.

Тип результата int

property name

Имя атрибута.

Тип результата str

property precision

Точность.

Тип результата int

property type_string

Тип в виде строки без длины и точности.

Тип результата str

property typedef

Описание типа.

Строка вида `<тип>[:длина][.точность]`.

Тип результата str

Схема таблицы - Schema

class axipy.da.**Schema**(**attributes*, *coordsystem=None*)

Схема таблицы. Представляет собой список атрибутов. Организован в виде `list` и свойством *coordsystem*. При задании *coordsystem* соответственно создается геометрический атрибут.

Параметры

- **attributes** (`List`[*Attribute*]) – Список атрибутов.
- **coordsystem** (`Union`[`str`, *CoordSystem*, `None`]) – Система координат для геометрического атрибута. Может быть задана или в виде строки (подробнее *axipy.cs.CoordSystem.from_string()*) или как объект СК.

Имеет стандартные функции работы со списком.

Список 3: Пример использования

```
count = len(schema)  # количество атрибутов
attribute = schema[2]  # читает
schema[2] = attr.string('attr', 30)  # изменяет
del schema[2]  # удаляет
schema.append(attr.string('new_attr'))  # добавляет в конец
schema.insert(2, attr.integer('num_attr'))  # добавляет по индексу
index = schema.index('new_attr')  # ищет по имени
assert 'new_attr' in schema  # проверяет существование
schema.remove('new_attr')  # удаляет по имени
```

См.также:

axipy.da.Attribute, *axipy.da.AttributeFactory()*.

attribute_names()

Возвращает список имен атрибутов.

Тип результата `list`

property coordsystem

Система координат.

Тип результата `Optional`[*CoordSystem*]

Результат None, если СК отсутствует.

См.также:

axipy.da.Table.is_spatial

Список 4: Пример использования

```
if schema.coordsystem is not None:  # проверяет существование
    pass
crs_string = schema.coordsystem  # получает
schema.coordsystem = 'epsg:4326'  # изменяет
schema.coordsystem = None  # удаляет
```

insert(*index*, *attr*)

Вставляет атрибут.

Параметры

- **index** (int) – Индекс, по которому производится вставка.
- **attr** (*Attribute*) – Атрибут.

Фабрика атрибутов - AttributeFactory

class axipy.da.**AttributeFactory**

Фабричный класс создания атрибутов и схем таблиц.

DEFAULT_STRING_LENGTH

Длина символьных атрибутов по умолчанию.

Type int

DEFAULT_DECIMAL_LENGTH

Длина десятичных атрибутов по умолчанию.

Type int

DEFAULT_DECIMAL_PRECISION

Точность десятичных атрибутов по умолчанию.

Type int

Пример создания атрибута:

```
string_attr = attr.string('attr_name', 80)
```

Пример создания схемы:

```
schema = attr.schema(
    attr.string('column_1'),
    attr.integer('column_2'),
    coordsystem='prj:Earth Projection 12, 62, "m", 0'
)
```

Примечание: Используйте более короткий псевдоним этого класса `axipy.da.attr`.

См.также:

axipy.da.Attribute, *axipy.da.Schema*.

static bool(*name*)

Создает атрибут логического типа.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static date(*name*)

Создает атрибут типа дата.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static datetime(*name*)

Создает атрибут типа дата и время.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static decimal(*name*, *length=10*, *precision=5*)
Создает атрибут десятичного типа.

Параметры

- **name** (str) – Имя атрибута.
- **length** (int) – Длина атрибута. Количество символов, включая запятую.
- **precision** (int) – Число знаков после запятой.

Тип результата *Attribute*

static double(*name*)
Создает атрибут вещественного типа.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static float(*name*)
Создает атрибут вещественного типа.

То же, что и *double()*

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static integer(*name*)
Создает атрибут целого типа.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

static schema(**attributes*, *coordsystem=None*)
Создает схему из списка атрибутов.

Параметры

- ***attributes** – Список атрибутов.
- **coordsystem** (Optional[str]) – Система координат в виде строки.

Тип результата *Schema*

Результат Новая схема.

См.также:

axipy.cs.CoordSystem.from_string().

static string(*name*, *length=80*)
Создает атрибут строкового типа.

Параметры

- **name** (str) – Имя атрибута.
- **length** (int) – Длина атрибута.

Тип результата *Attribute*

static time(*name*)
Создает атрибут типа время.

Параметры **name** (str) – Имя атрибута.

Тип результата *Attribute*

Запись в таблице - Feature

class axipy.da.**Feature**(*properties={}*, *geometry=None*, *style=None*, *id=None*, ***kwargs*)

Запись в таблице.

Работа с записью похожа на работу со словарем dict. Но также допускает обращение по индексу.

Количество атрибутов:

```
count = len(feature)
```

Запись значения:

```
feature['my_attr_name'] = 'new_value'  # по ключу
feature[0] = 'another_value'  # по индексу
```

Чтение значения:

```
value = feature['my_attr_name']  # по ключу
another_value = feature[0]  # по индексу
```

Проверка наличия атрибута:

```
'my_attr_name' in feature  # по ключу
5 in feature  # по индексу
```

Значения атрибутов можно задать словарем или именованными аргументами.

Например:

```
feature = Feature({'name1': 'value1', 'name2': 'value2'})
# эквивалентно
feature = Feature(name1='value1', name2='value2')
```

Параметры

- **properties** (dict) – Значения атрибутов.
- **geometry** (Optional[*Geometry*]) – Геометрия.
- **style** (Optional[*Style*]) – Стиль.
- **id** (Optional[int]) – Идентификатор.
- ****kwargs** – Значения атрибутов.

property geometry

Геометрия записи.

Например:

```
point = feature.geometry
# ...
feature.geometry = new_point
```

См.также:

Feature.has_geometry(), GEOMETRY_ATTR

Тип результата Optional[*Geometry*]

Результат Значение геометрического атрибута; или None, если значение пустое или отсутствует.

get(*key*, *default=None*)
Возвращает значение заданного атрибута.

Параметры

- **key** (str) – Имя атрибута.
- **default** (Optional[Any]) – Значение по-умолчанию.

Результат Искомое значение, или значение по-умолчанию, если заданный атрибут отсутствует.

has_geometry()
Проверяет, имеет ли запись атрибут с геометрией.

Тип результата bool

has_style()
Проверяет, имеет ли запись атрибут со стилем.

Тип результата bool

property id
Идентификатор записи в таблице.

Несохраненные записи в таблице будут иметь отрицательное значение.

См.также:

Table.is_modified, *Table.commit()*

Тип результата int

Результат 0 если идентификатор не задан.

items()
Возвращает список пар имя - значение.

Например:

```
for key, value in feature.items():
    pass
```

Тип результата List[tuple]

keys()
Возвращает список имен атрибутов.

Тип результата List[str]

property style
Стиль записи.

Например:

```
style = feature.style
# ...
feature.style = style
```

См.также:

Feature.has_style(), STYLE_ATTR

Тип результата Optional[*Style*]

Результат Значение атрибута со стилем; или None, если значение пустое или отсутствует.

to_geojson()
Представляет запись в виде, похожем на „GeoJSON“.

Тип результата dict

values()
Возвращает список значений атрибутов.

Тип результата List

Растр - Raster

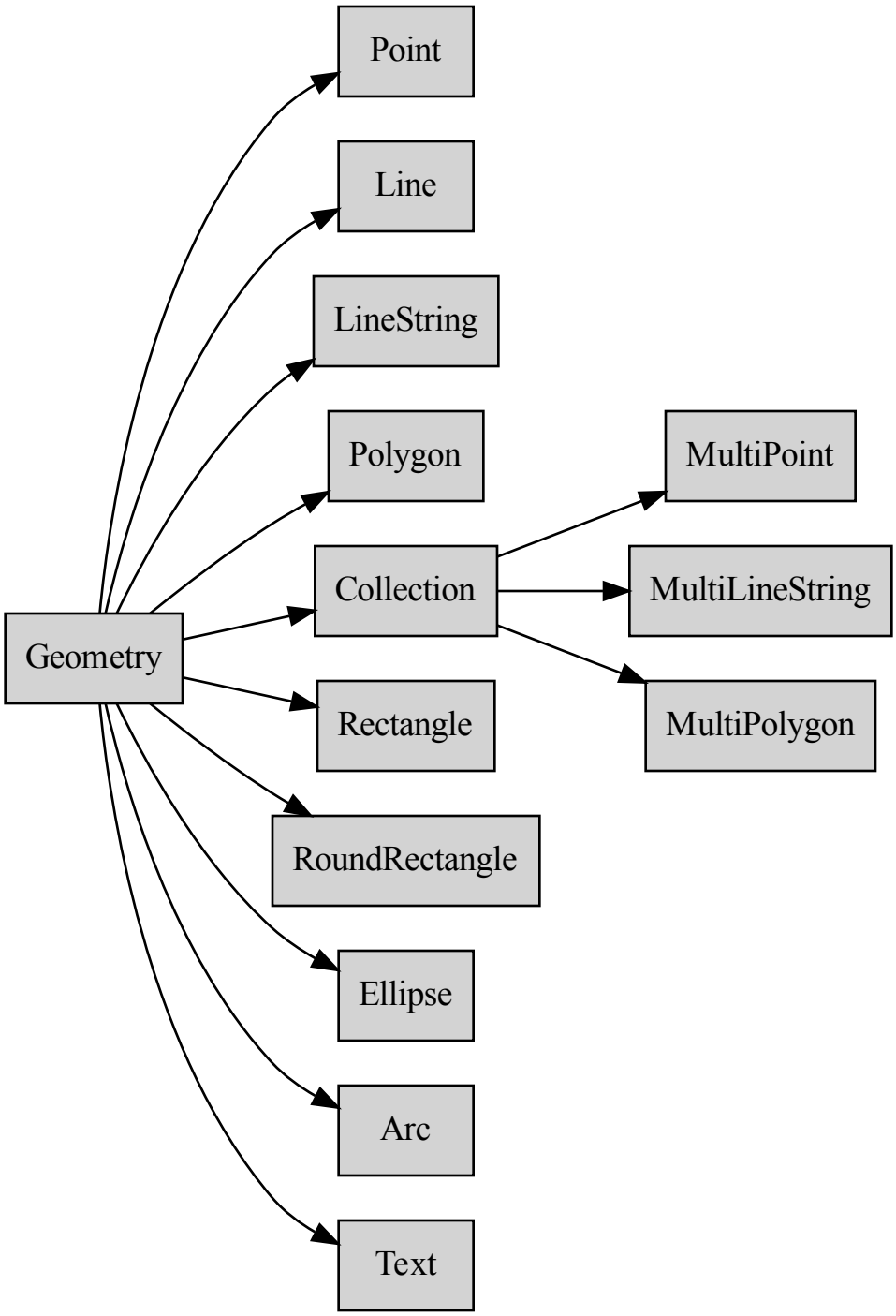
class axipy.da.**Raster**

Базовые классы: *axipy.da.DataObject*

Растровый объект.

12.1.5 Геометрия - Geometry

Иерархия геометрических классов:



class axipy.da.**Geometry**

Абстрактный класс геометрического объекта (геометрии).

affine_transform(*trans*)

Трансформирует объект исходя из заданных параметров трансформации.

См.также:

Для простых операций типа сдвиг, масштабирование и поворот, рекомендуется использовать *shift()*, *scale()*, *rotate()* соответственно.

Параметры **trans** (QTransform) – Матрица трансформации.

Тип результата *Geometry*

almost_equals(*other*, *tolerance*)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** (*Geometry*) – Сравниваемый объект.
- **tolerance** (float) – Точность сравнения.

Тип результата bool

Результат Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary()

Возвращает границы геометрии в виде полилинии.

Тип результата *Geometry*

property bounds

Возвращает минимальный ограничивающий прямоугольник.

Тип результата *Rect*

buffer(*distance*, *resolution=16*, *capStyle=1*, *joinStyle=1*, *mitreLimit=5.0*)

Производит построение буфера.

Параметры

- **distance** (float) – Ширина буфера.
- **resolution** (int) – Количество сегментов на квадрант.
- **capStyle** (int) – Стиль окончания.
- **joinStyle** (int) – Стиль соединения.
- **mitreLimit** (float) – Предел среза.

Тип результата *Geometry*

centroid()

Возвращает центроид геометрии.

Тип результата *Geometry*

clone()

Создает копию объекта.

Тип результата *Geometry*

contains(*other*)
Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

Тип результата bool

convex_hull()
Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

Тип результата *Geometry*

property coordsystem
Система Координат (СК) геометрии.

Тип результата *CoordSystem*

covers(*other*)
Возвращает True, если геометрия охватывает геометрию *other*.

Тип результата bool

crosses(*other*)
Возвращает True, если при пересечении геометрий объекты частично пересекаются.

Тип результата bool

difference(*other*)
Возвращает область первой геометрии, которая не пересечена второй геометрией.

Тип результата *Geometry*

disjoint(*other*)
Возвращает True, если геометрии не пересекаются и не соприкасаются.

Тип результата bool

envelope()
Возвращает полигон, описывающий заданную геометрию.

Тип результата *Geometry*

equals(*other*)
Производит сравнение с другой геометрией.

Параметры **other** (*Geometry*) – Сравниваемая геометрия.

Тип результата bool

Результат Возвращает True, если геометрии равны.

static from_wkb(*wkb*, *coordsystem=None*)
Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** (bytes) – Строка WKB
- **coordsystem** (Optional[*CoordSystem*]) – Система координат, которая будет установлена для геометрии.

Пример:

```
wkb = b'$@$@'
pnt = Geometry.from_wkb(wkb)
print(pnt.wkt)
```

```
>>> POINT (10 10)
```

Тип результата *Geometry*

static from_wkt(*wkt*, *coordsystem=None*)

Создает геометрический объект из строки формата WKT.

Параметры

- **wkt** (str) – Строка WKT. Допустимо задание строки в формате с указанием SRID (EWKT). В данном случае система координат для создаваемой геометрии будет установлено исходя их этого значения.
- **coordsystem** (Optional[*CoordSystem*]) – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Пример:

```
from axipy import *
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

Тип результата *Geometry*

get_area(*u=None*)

Рассчитывает площадь, если объект площадной. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в квадратных метрах.

Пример:

```
# В проекции Меркатора:
csMercatorKm = CoordSystem. from_prj("10, 104, 1, 0")
poly = Polygon([(0,0), (0,2), (2, 2), (2,0)], cs=csMercatorKm)
print('Area Mercator km:', poly.get_area())
print('Area Mercator m:', poly.get_area(unit.sq_m))
print('Perimeter Mercator km:', poly.get_perimeter())
print('Perimeter Mercator m:', poly.get_perimeter(unit.m))
# В Широте/Долготе
poly.coordsystem = CoordSystem.from_prj("1, 104")
print('Area LL m:', poly.get_area())
print('Area LL km:', poly.get_area(unit.sq_km))
print('Perimeter LL m:', poly.get_perimeter())
print('Perimeter LL km:', poly.get_perimeter(unit.km))

>>> Area Mercator km: 4.017946986632519
>>> Area Mercator m: 4017946.9866325194
>>> Perimeter Mercator km: 8.017972048421552
>>> Perimeter Mercator m: 8017.972048421552
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
```

Параметры **u** (Optional[*AreaUnit*]) – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

Тип результата float

get_distance(*other*, *u=None*)

Производит расчет расстояния до объекта *other*. Результат возвращает в СК текущего объекта.

Параметры

- **other** (*Geometry*) – Анализируемый объект.
- **u** (Optional[*LinearUnit*]) – Единицы измерения, в которых требуется получить результат.

Пример:

```
# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))
#Установим разные СК для точек
first.coordsystem = CoordSystem.from_prj("10, 104, 7, 0")
second.coordsystem = CoordSystem.from_prj("1, 104")
print('На сфере м:', first.get_distance(second))
print('На сфере км:', first.get_distance(second, unit.km))

>>> В плане: 10.0
>>> На сфере м: 1111948.7428468117
>>> На сфере км: 1111.9487428468117
```

Тип результата float

get_length(*u=None*)

Рассчитывает длину геометрии. Метод применим только для линейных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример:

```
# В проекции Меркатора:
csMercatorKm = CoordSystem. from_prj("10, 104, 1, 0")
ls = Line((0,0), (10,0), csMercatorKm)
print('Length Mercator km:', ls.get_length())
print('Lenght Mercator m:', ls.get_length(unit.m))
# В Широте/Долготе
csLL = CoordSystem.from_prj("1, 104")
ls = Line((0,0), (10,0), csLL)
print('Length LL m:', ls.get_length())
print('Length LL km:', ls.get_length(unit.km))

>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
```

Параметры **u** (Optional[*LinearUnit*]) – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

Тип результата float

get_perimeter(*u=None*)
Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В противном случае возвращает 0. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. *get_area()*

Параметры **u** (Optional[*LinearUnit*]) – Единица измерения, в которой необходимо получить результат. Если не задана, то используется единица измерения для СК. Если и она не задана, то производится расчет на плоскости.

Тип результата float

intersection(*other*)
Возвращает область пересечения с другой геометрией.

Тип результата *Geometry*

intersects(*other*)
Возвращает True, если геометрии пересекаются.

Тип результата bool

property is_valid
Проверяет геометрию на валидность.

Тип результата bool

property is_valid_reason
Если геометрия неправильная, возвращает краткую аннотацию причины.

Тип результата str

property name
Возвращает наименование геометрического объекта.

Тип результата str

overlaps(*other*)
Возвращает True, если пересечение геометрий отличается от обеих геометрий.

Тип результата bool

relate(*other*)
Проверяет отношения между объектами. Подробнее см. DE-9IM

Тип результата str

reproject(*cs*)
Перепроецирует геометрию в другую систему координат.

Параметры **cs** (*CoordSystem*) – СК, в которой требуется получить объект.

Тип результата *Geometry*

rotate(*point*, *angle*)
Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** (Union[*Pnt*, Tuple[float, float]]) – Точка, вокруг которой необходимо произвести поворот.
- **angle** (float) – Угол поворота в градусах.

Тип результата *Geometry*

scale(*kx*, *ky*)
Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** (float) – Масштабирование по координате X.
- **ky** (float) – Масштабирование по координате Y.

Тип результата *Geometry*

shift(*dx*, *dy*)
Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** (float) – Сдвиг по координате X.
- **dy** (float) – Сдвиг по координате Y.

Тип результата *Geometry*

symmetric_difference(*other*)
Возвращает логический XOR областей геометрий (объединение разниц).

Параметры **other** (*Geometry*) – Геометрия для анализа.

Тип результата *Geometry*

touches(*other*)
Возвращает True, если геометрии соприкасаются.

Тип результата bool

property type
Возвращает тип геометрического элемента.

Таблица 2: Возможные значения

Значение	Наименование
Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
MultiGeometry	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectange	Прямоугольник
RoundedRectange	Скругленный прямоугольник
Text	Текст

Пример:

```
point = Geometry.from_wkt('POINT (10 10)')
if point.type == GeometryType.Point:
    print('Это точка')
```

Тип результата GeometryType

union(*other*)
Возвращает результат объединения двух геометрий.

Тип результата *Geometry*

within(*other*)
Возвращает True, если геометрия находится полностью внутри геометрии *other*.

Тип результата bool

property wkb
Возвращает WKB строку для геометрии.

Тип результата bytes

property wkt
Возвращает WKT строку для геометрии.

Тип результата str

12.1.5.1 Точечный объект - Point

class axipy.da.**Point**(*x*, *y*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа точка.

Параметры

- **x** (float) – X координата

- **y** (float) – Y координата
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
cs = CoordSystem.from_prj("1, 104")
p = Point(23, 45, cs) # Создадим точку.
p.x = 55 # Изменим значение координаты X
```

property x
X Координата.

Тип результата float

property y
Y Координата.

Тип результата float

12.1.5.2 Линия - Line

class axipy.da.**Line**(*begin*, *end*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа линия.

Параметры

- **begin** (*Pnt*) – Начальная точка линии.
- **end** (*Pnt*) – Конечная точка линии.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
cs = CoordSystem.from_prj("1, 104")
line = Line(Pnt(22, 44), (100, 101), cs) # Создадим линию с различным подходом␣
↪при указании начальной о конечной точки
line.begin = (88, 99) # Заменим координаты начальной точки.
line.end = Pnt(120, 120)
```

property begin
Начальная точка линии. Точки допустимо создавать как экземпляр Pnt либо в виде пары „float“ значений tuple

Тип результата *Pnt*

property end
Конечная точка линии.

Тип результата *Pnt*

12.1.5.3 Полилиния - LineString

class axipy.da.**LineString**(**points*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа полилиния.

Параметры

- **points** (Union[*Pnt*, Tuple[float, float]]) – Список точек. Может задаваться следующим образом:
  - В виде списка list из пар tuple.
  - В виде перечня точек. В данном случае, если необходимо задать СК, то требуется явно указать наименование параметра.
  - В виде итератора по элементам, состоящих из пар tuple.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
ls = LineString([(1, 2), Pnt(3, 4), Pnt(5, 6), (7, 8)]) # Создадим полилинию без␣
↪СК
ls.points[1] = (33, 44) # Обновим точку с индексом 1. Допустимо только обновление␣
↪точки целиком. Изменение координат по одиночке не поддерживается.
ls.points.append((9,10)) # Добавим точку в конец
ls.points.remove(2) # Удалим вторую точку
ls.points.insert(3, (11,12)) # Добавим точку на позицию 3
for p in ls.points: # Просмотр всех точек
    print("point:", p)
ls2 = LineString((1, 2), (3, 4), (5, 6), (7, 8), cs=csLatLon) # Создание полинии,␣
↪передав перечень точек.
itr = (a for a in ls.points) # Создадим итератор на базе точек первой полилинии
ls3 = LineString(itr) #
```

property points

Точки полилинии. Реализован как список python list точек Pnt. Также поддерживаются список пар tuple.

Примечание: При обновление значения точки допустимо только изменение ее заменой.

Тип результата List[*Pnt*]

12.1.5.4 Полигон - Polygon

class axipy.da.**Polygon**(**points*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа полигон. Представляет собой часть плоскости, ограниченной замкнутой полилинией. Кроме внешней границы, полигон может иметь одну или несколько внутренних (дырок).

Параметры

- **points** (Union[*Pnt*, Tuple[float, float]]) – Список точек внешнего контура. Может задаваться следующим образом:
  - В виде списка list из пар tuple.
  - В виде перечня точек. В данном случае, если необходимо задать СК, то требуется явно указать наименование параметра.
  - В виде итератора по элементам, состоящих из пар tuple.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
poly = Polygon([(0, 0), Pnt(1, 10), Pnt(10, 11), (10, 2)]) # Создадим объект
poly.points[2] = (14, 15) # Поменяем вторую точку
poly.points.insert(3, (11, 5)) # Добавим точку
for p in poly.points: # Просмотр точек полигона
    print("point:", p)
poly.points.remove(2) # Удалим вторую точку
```

static from_rect(*rect*, *cs=None*)

Создает полигон на базе прямоугольника.

Параметры

- **rect** (*Rect*) – Прямоугольник, на основе которого формируются координаты.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Тип результата *Polygon*

property holes

Дырки полигона. Реализован в виде списка list.

Пример:

```
poly.holes.append([(2,2), (2,4), (5,3)]) # Добавим дырку
for p in poly.holes[0]: # Просмотр точек дырки полигона
    print("Point of hole:", p)
print('Вторая точка первой дырки:', poly.holes[0][1])
poly.holes[0][1] = (33,44) # Обновим значение этой точки
```

Тип результата List[*Pnt*]

property points
Точки полигона. Реализован как список python `list` точек `Pnt`. Также поддерживаются список пар `tuple`.

Тип результата `List`[*Pnt*]

12.1.5.5 Коллекция геометрий - Collection

class axipy.da.**Collection**(*cs=None*)

Базовые классы: *axipy.da.Geometry*

Коллекция разнотипных геометрических объектов. Допустимо хранение геометрических объектов различного типа, за исключением коллекций. Доступ к элементам производится по аналогии работы со списком `list`.

Параметры cs (`Optional`[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Для получения размера коллекции используйте функцию len:

```
cnt = len(coll)
```

Доступ к элементам производится по индексу. Нумерация начинается с 0.

В качестве примера получим первый элемент:

```
coll[1]
```

Обновление геометрии в коллекции так же производится по ее индексу. Так же допустимо изменение некоторых свойств геометрии в зависимости от ее типа.

Примеры установки элемента с индексом 1:

```
coll[1] = (2,2) # Как точки с координатами (2,2)
coll[1] = [(101, 102), (103, 104), (105, 106)] # Как полилинии
coll[1] = Polygon((101, 102), (103, 104), (105, 106)) # Как полигона
```

append(**value*)
Добавление геометрии в коллекцию. Задание параметров аналогично указанию их при создании объектов конкретного типа. Если опустить указание класса, то для одной точки или пары значений float будет создан точечный объект *Point*, если точек больше, - объект класса *LineString*.

Примеры:

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = Collection()
coll.append((1,2)) # Точка
coll.append(1,2) # Точка
coll.append([(3,4), (5, 5), (10, 0)]) # Полилиния в виде :class:`list`. Можно
↪это сделать через конструктор :class:`LinearString`.
coll.append((3,4), (5, 5), (10, 0)) # Полилиния
coll.append(Polygon([(3,4), (5, 5), (10, 0)])) # Полигон
```

remove(*idx*)
» Удаление геометрии из коллекции.

Параметры idx (`int`) – Индекс геометрии в коллекции.

12.1.5.6 Коллекция точек - MultiPoint

class axipy.da.**MultiPoint**(*cs=None*)

Базовые классы: *axipy.da.Collection*

Коллекция точечных объектов. Может содержать только объекты типа точка.

Параметры **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
mpoint = MultiPoint() # Создаем коллекцию.
# Добавим точку разными способами.
p = Point(23, 34)
mpoint.append(p)
mpoint.append((12, 12))
mpoint.append(Pnt(10,10))
mpoint[0] = (66,66) # Заменяем первый объект (индекс 0)
mpoint[0].x = 77 # Заменим только координату x для первого объекта в коллекции
mpoint.remove(1) # Удаляем второй объект
```

12.1.5.7 Коллекция полилиний - MultiLineString

class axipy.da.**MultiLineString**(*cs=None*)

Базовые классы: *axipy.da.Collection*

Коллекция полилиний. Может содержать только объекты типа полилиния.

Параметры **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
msl = MultiLineString() # Создадим саму коллекцию.
ls = LineString([(1, 2), (3, 4), (5, 6), (7, 8)])
msl.append(ls) # Добавим как объект по ссылке
msl.append(LineString([(11, 12), (13, 14), (15, 16)])) # Добавим как объект
msl.append([(21, 22), (23, 24), (25, 26)]) # Добавим как перечень точек как␣
↪:class:`list`
msl[2].points[1] = (101, 102) # Обновим значение точки 3 полилинии по индексу 2.
msl.remove(0) # Удалим первый объект из коллекции.
msl[1].points.remove(2) # Удалим точку с индексом 1 из полилинии 2
msl[0] = [(101, 102), (103, 104), (105, 106), (107, 108)] # Обновим первую␣
↪геометрию
```

12.1.5.8 Коллекция полигонов - MultiPolygon

class axipy.da.**MultiPolygon**(*cs=None*)

Базовые классы: *axipy.da.Collection*

Коллекция полигонов. Может содержать только объекты типа полигон.

Параметры **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
mpoly = MultiPolygon() # Создадим саму коллекцию.
mpoly.append([(1, 2), (3, 4), (5, 6), (7, 8)]) # Добавим полигон в виде списка␣
↪точек
mpoly.append(poly) # Добавим ранее созданный с дыркой (пример см.␣
↪:class:`Polygon`)
mpoly[1].holes[0][1] = (99,99) # Изменение второй точки дырки
mpoly[1].points[0] = (0, 0) # Заменим первую (она же последняя) точку полигона
poly2 = Polygon([(11, 12), (13, 14), (15, 16), (17, 18)])
mpoly[0] = poly2 # Полностью заменим первый полигон
```

12.1.5.9 Прямоугольник - Rectangle

class axipy.mi.**Rectangle**(**par, cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа прямоугольник.

Параметры

- **par** (Union[*Rect*, float]) – Прямоугольник класса Rect или перечень координат через запятую (*xmin, ymin, xmax, ymax*) или списком list.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
r1 = Rectangle(0, 0, 40, 20) # Создадим объект через перечень координат.
r2 = Rectangle(Rect(0, 0, 40, 20)) # Создадим объект передав объект :class:`Rect`.
r3 = Rectangle([0, 0, 40, 20]) # Создадим объект передав списком :class:`list`.
```

property xmax
Максимальное значение X.

Тип результата float

property xmin
Минимальное значение X.

Тип результата float

property ymax
Максимальное значение Y.

Тип результата float

property ymin
Минимальное значение Y.

Тип результата float

12.1.5.10 Скругленный прямоугольник - RoundRectangle

class axipy.mi.**RoundRectangle**(*rect*, *xRad*, *yRad*, *cs=None*)

Базовые классы: *axipy.mi.Rectangle*

Геометрический объект типа скругленный прямоугольник.

Параметры

- **rect** (Union[*Rect*, list]) – Прямоугольник класса Rect или как list.
- **xRad** (float) – Скругление по X.
- **yRad** (float) – Скругление по Y.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
r1 = RoundRectangle(Rect(0,0,22,33), 0.1, 0.1)
r2 = RoundRectangle([0,0,22,33], 0.1, 0.1)
```

property xRadius
Скругление углов по координате X.

Тип результата float

property yRadius
Скругление углов по координате Y.

Тип результата float

12.1.5.11 Эллипс - Ellipse

class axipy.mi.**Ellipse**(*rect*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа эллипс.

Параметры

- **rect** (Union[*Rect*, list]) – Прямоугольник класса Rect или как list.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
e1 = Ellipse(Rect(0,0,22,33))
e2 = Ellipse([0,0,22,33])
e1.center = (10,10) # Переопределим центр
e1.majorSemiAxis = 10 # Задание большой полуоси
e1.minorSemiAxis = 5 # Задание малой полуоси
```

property center
Центр эллипса.

Тип результата *Pnt*

property majorSemiAxis
Радиус большой полуоси эллипса.

Тип результата float

property minorSemiAxis
Радиус малой полуоси эллипса.

Тип результата float

12.1.5.12 Дуга - Arc

class axipy.mi.**Arc**(*rect*, *startAngle*, *endAngle*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа дуга.

Параметры

- **rect** (Union[*Rect*, list]) – Прямоугольник класса Rect или как list.
- **startAngle** (float) – Начальный угол дуги.
- **endAngle** (float) – Конечный угол дуги.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
a1 = Arc(Rect(0,0,22,33), 0, 90)
a2 = Arc([0,0,22,33], 0, 90)
```

property center
Центр дуги.

Тип результата *Pnt*

property endAngle
Конечный угол дуги.

Тип результата float

property startAngle
Начальный угол дуги.

Тип результата float

property xRadius
Радиус большой полуоси прямоугольника, в который вписана дуга.

Тип результата float

property yRadius
Радиус малой полуоси прямоугольника, в который вписана дуга.

Тип результата float

12.1.5.13 Текст - Text

class axipy.mi.**Text**(*text*, *point*, *cs=None*)

Базовые классы: *axipy.da.Geometry*

Геометрический объект типа текст.

Предупреждение: Геометрия текста, и, в отличие от остальных типов объектов, определяется так же и стилем его оформления *axipy.da.TextStyle*.

Параметры

- **text** (str) – Текст
- **topLeft** – Точка привязки. Допустимо задание точки Pnt или как пара tuple.
- **cs** (Optional[*CoordSystem*]) – Система Координат, в которой создается геометрия.

Пример:

```
t1 = Text("Пример", (10, 10))
t2 = Text("Пример", Pnt(10, 10))
```

property angle
Угол поворота текста.

Тип результата float

property startPoint
Координаты точки привязки.

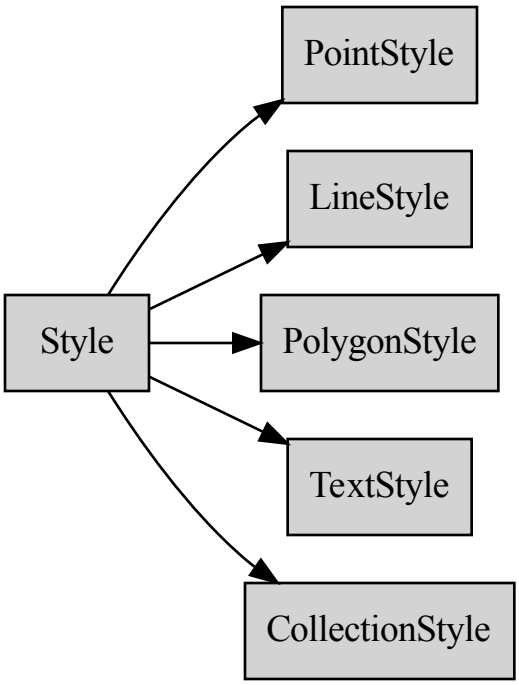
Тип результата *Pnt*

property text
Текст.

Тип результата str

12.1.6 Стиль - Style

Иерархия классов стилей геометрических объектов:



class axipy.da.**Style**

Абстрактный класс стиля оформления геометрического объекта.

Определяет как будет отрисован геометрический объект.

classmethod for_geometry(*geom*)

Возвращает стиль по умолчанию для переданного объекта.

Параметры geom (*Geometry*) – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

Тип результата *Style*

classmethod from_mapinfo(*mapbasic_string*)

Получает стиль из строки формата MapBasic.

Параметры mapbasic_string (str) – Строка в формате MapBasic.

Пример:

```
style = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255)")
```

Тип результата *Style*

to_mapinfo()
Возвращает строковое представление в формате MapBasic.

Пример:

```
style.to_mapinfo()

>>> Pen (1, 2, 0) Brush (8, 255)
```

Тип результата str

12.1.6.1 Стиль точек - PointStyle

class axipy.da.**PointStyle**

Базовые классы: *axipy.da.Style*

Стиль оформления точечных объектов.

По умолчанию создается стиль на базе шрифта True Type, а параметры аналогичны значениям по умолчанию в методе *create_mi_font()*.

Поддерживается 3 вида оформления:

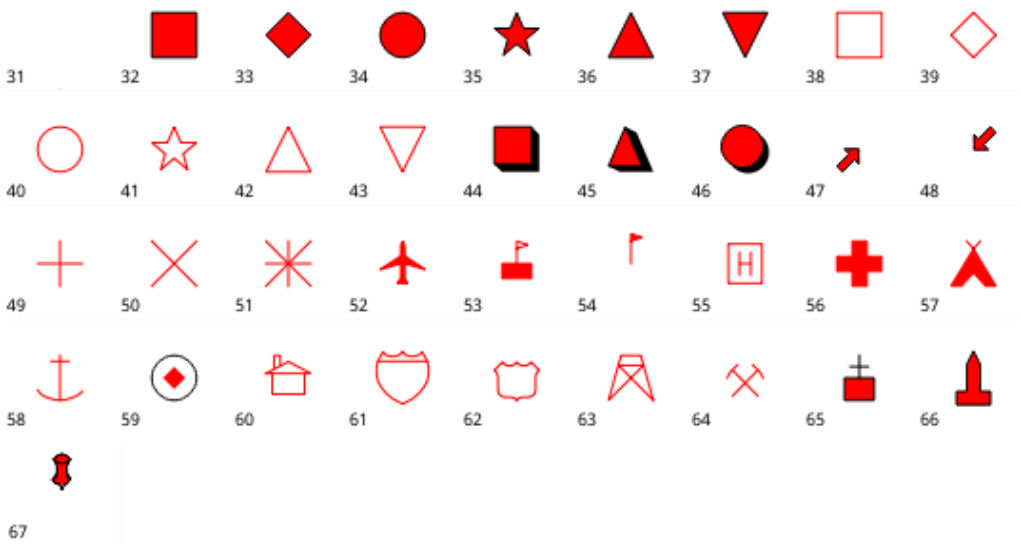
- Совместимое с MapInfo версии 3. Для создания такого стиля необходимо использовать *create_mi_compat()*.
- На базе шрифтов True Type. Задано по умолчанию. Стиль создается посредством *create_mi_font()*.
- На базе растрового файла. Стиль можно создать с помощью *create_mi_picture()*.

static create_mi_compat(*symbol=35*, *color=PySide2.QtCore.Qt.GlobalColor.red*, *pointSize=8*)
Создание стиля в виде совместимого с MapInfo 3.

Параметры

- **symbol** (int) – Номер символа, который будет отображен при отрисовке. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67,
- **color** (QColor) – Цвет символа
- **pointSize** (int) – Целое число, размер символа в пунктах от 1 до 48.

В системе доступны следующие стили:



Подробнее: Стиль MapInfo

Тип результата *PointStyle*

static create_mi_font(*symbol=35*, *color=PySide2.QtCore.Qt.GlobalColor.red*, *size=8*, *fontname='Axioma MI MapSymbols'*, *fontstyle=0*, *rotation=0.0*)

Создание стиля на базе шрифта True Type.

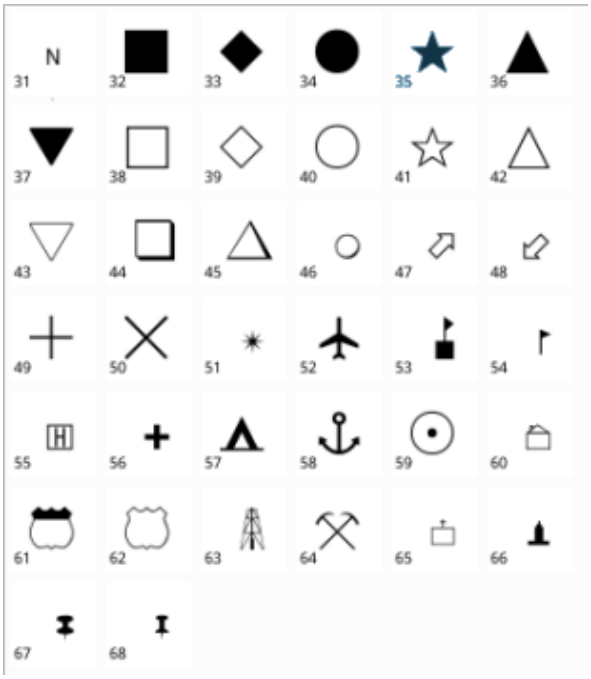
Параметры

- **symbol** (int) – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** (QColor) – Цвет символа
- **pointSize** – Целое число, размер символа в пунктах от 1 до 48;
- **fontname** (str) – Строка с именем шрифта TrueType (например, значение по умолчание „Axioma MI MapSymbols“)
- **fontstyle** (int) – Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** (float) – Угол поворота символа в градусах.

Таблица 3: Возможные значения параметра *fontstyle*

Значение	Наименование
0	Обычный текст
1	Жирный текст
16	Черная кайма вокруг символа
32	Тень
256	Белая кайма вокруг символа

В системе доступны следующие стили:



Подробнее: Стиль True Type

Тип результата *PointStyle*

static create_mi_picture(*filename*, *color=PySide2.QtCore.Qt.GlobalColor.black*, *size=12*, *customstyle=0*)

Создание стиля с ссылкой на растровый файл.

Параметры

- **filename** (str) – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находится в каталоге CustSymb с ресурсами. Например, *“Arrow.BMP”*.
- **color** (QColor) – Цвет символа.
- **size** (int) – Размер символа в в пунктах от 1 до 48.
- **customstyle** (int) – Задание дополнительных параметров стиля оформления.

Таблица 4: Возможные значения параметра *customstyle*

Значение	Наименование
0	Флажки Фон и Покрасить одним цветом не установлены. Символ показывается стандартно. Все белые точки изображения становятся прозрачными и под ними видны объекты Карты.
1	Установлен флажок Фон; все белые точки изображения становятся непрозрачными.
2	Установлен флажок Покрасить одним цветом все не белые точки изображения красятся в цвет символа.
3	Установлены флажки Фон и Покрасить одним цветом.

Подробнее: Стиль растрового символа

Тип результата *PointStyle*

12.1.6.2 Стиль линий - LineStyle

class axipy.da.**LineStyle**(*pattern=2, color=PySide2.QtCore.Qt.GlobalColor.black, width=1*)

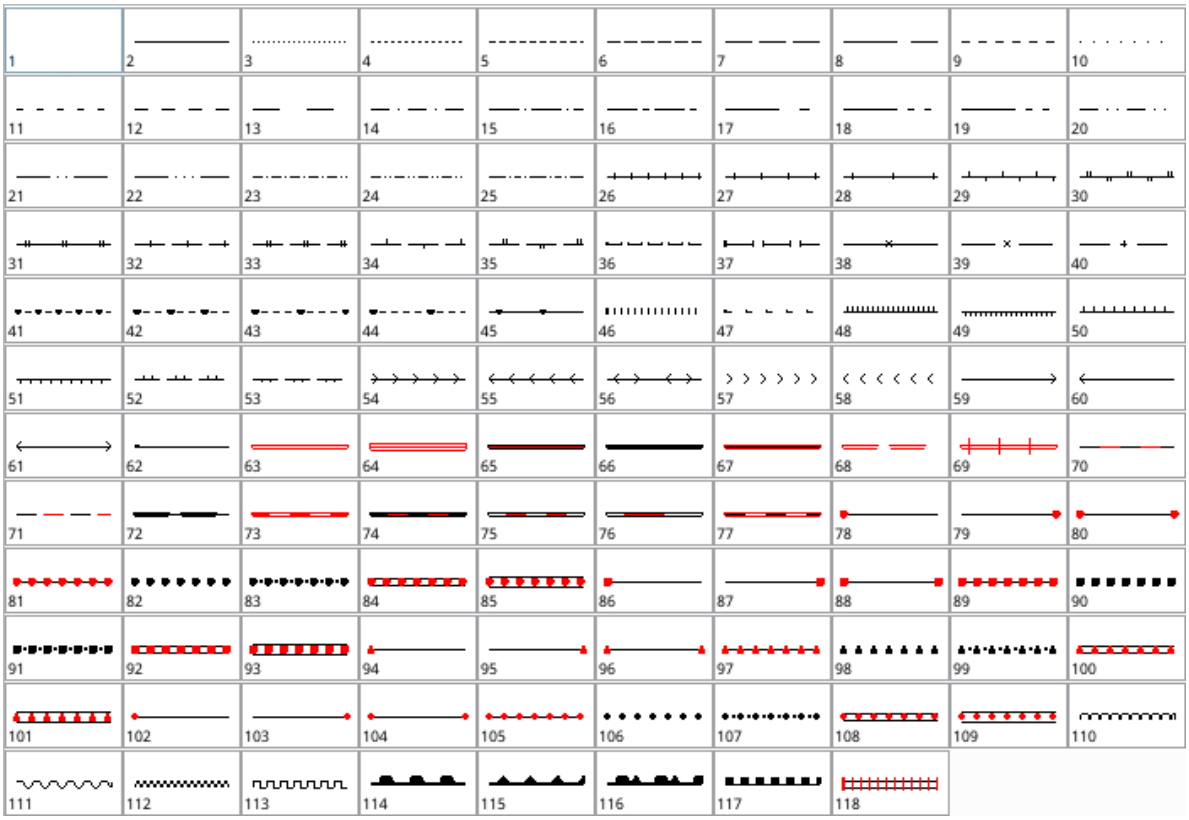
Базовые классы: *axipy.da.Style*

Стиль линейного объекта, совместимый с MapInfo.

Параметры

- **pattern** (int) – Тип линии. Типы линий обозначаются кодами от 1 до 118. Тип 1 представляет собой невидимую линию.
- **color** (QColor) – Цвет линии
- **width** (int) – Толщина линии. Задается числом от 0 до 7, при этом линия нулевой ширины невидима на экране. 11-2047 - это значения, которые могут быть преобразованы в пункты: ширина линии = (число пунктов * 10) + 10 Значение 0 допустимо только для типа линии 1 или невидимых линий.

В системе доступны следующие стили линии:



Подробнее: Стиль линейных объектов

12.1.6.3 Стиль полигонов - PolygonStyle

class axipy.da.**PolygonStyle**(*pattern=1*, *color=PySide2.QtCore.Qt.GlobalColor.white*, *pattern_pen=1*)

Базовые классы: *axipy.da.Style*

Стиль площадного объекта. По умолчанию создается прозрачный стиль с черной окантовкой.

Пример:

```
# Создадим стиль по умолчанию
plstyle = PolygonStyle()
print(plstyle.to_mapinfo())
# Назначим цвет заливки и фона заливки
plstyle.set_brush(color=Qt.green, bgColor=Qt.blue)
print(plstyle.to_mapinfo())
# Установим обводку
plstyle.set_pen(color=Qt.black)
print(plstyle.to_mapinfo())

>>> Brush (1, 16777215)
>>> Brush (1, 65280, 255)
>>> Pen (1, 2, 0)  Brush (1, 65280, 255)
```

Параметры

- **pattern** (int) – Номер стиля заливки.
- **color** (QColor) – Цвет основной заливки.
- **pattern_pen** (int) – Цвет обводки. По умолчанию обводка отсутствует.

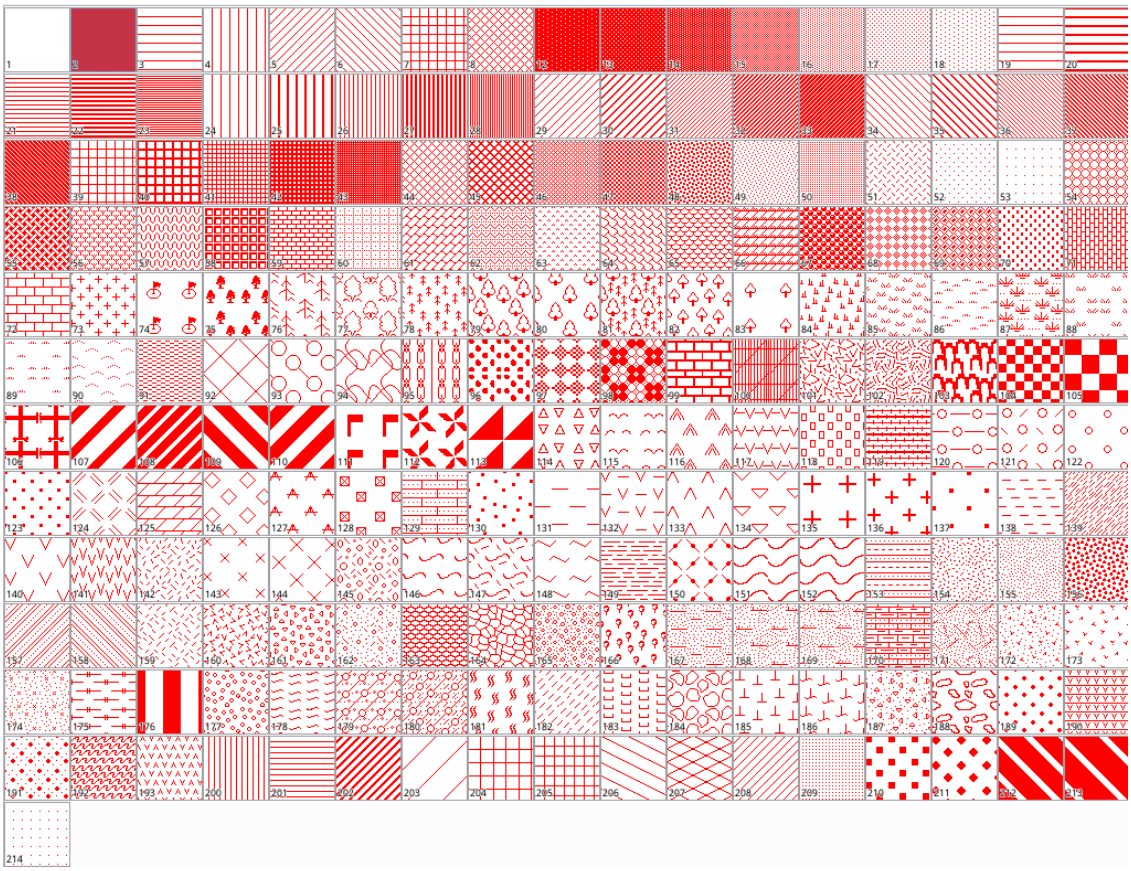
set_brush(*pattern=1*, *color=PySide2.QtCore.Qt.GlobalColor.white*, *bgColor=PySide2.QtCore.Qt.GlobalColor.transparent*)

Задание стиля заливки площадного объекта.

Параметры

- **pattern** (int) – Номер стиля заливки. Шаблон задается числом от 1 до 71, при этом в шаблоне с номером 1 оба цвета отсутствуют, а в шаблоне 2 отсутствует цвет фона. Шаблоны с кодами 9-11 зарезервированы для внутренних целей.
- **color** (QColor) – Цвет основной заливки.
- **bgColor** (QColor) – Цвет заднего фона, если заливка неполная.

В системе доступны следующие стили заливки:



Подробнее: Стиль заливки полигона

set_pen(*pattern=2*, *color=PySide2.QtCore.Qt.GlobalColor.black*, *width=1*)

Задание стиля обводки. Параметры аналогичны при задании стиля линии *LineStyle()*

Параметры

- **pattern** (int) – Номер стиля линии.
- **color** (QColor) – Цвет линии
- **width** (int) – Толщина линии.

12.1.6.4 Стиль текста - TextStyle

class axipy.da.**TextStyle**(*fontname*, *size*, *style=0*, *forecolor=PySide2.QtCore.Qt.GlobalColor.black*, *backcolor=PySide2.QtCore.Qt.GlobalColor.transparent*)

Базовые классы: *axipy.da.Style*

Стиль текстового объекта.

Параметры

- **fontname** (str) – Наименование шрифта.
- **size** (int) – Размер шрифта в пунктах. Может принимать значение 0 для подписей в окне карты, так как они являются атрибутами карты и их размер определяется динамически.

- **style** (int) – Дополнительные параметры стиля. Подробнее см. в таблице ниже.
- **color** – Цвет шрифта
- **backcolor** (QColor) – Цвет заднего фона, если он задан.

Таблица 5: Возможные значения параметра style

Значение	Наименование
0	Обычный
1	Жирный
2	Курсив
4	Подчеркнутый
16	Контур (только для *Macintosh*)
32	Тень
256	Кайма
512	Капитель
1024	Разрядка

Подробнее: Стиль текста

12.1.6.5 Стиль коллекций - CollectionStyle

class axipy.da.**CollectionStyle**

Базовые классы: *axipy.da.Style*

Смешанный стиль для разнородного типа объектов.

Данный стиль представляет собой контейнер стилей. может применяться в купе с геометрическим объектом типа разнородная коллекция *axipy.da.Collection*. Для задания или переопределения стилей простейших объектов, необходимо вызывать соответствующие методы для необходимых типов объектов.

for_line(*style*)
Задание стиля для линейных объектов *LineStyle*.

for_point(*style*)
Задание стиля для точечных объектов *PointStyle*.

for_polygon(*style*)
Задание стиля для полигональных объектов *PolygonStyle*.

for_text(*style*)
Задание стиля для текстовых объектов *TextStyle*.

12.1.7 axipy.gui

Модуль пользовательского интерфейса.

В данном модуле содержатся классы связанные с пользовательским интерфейсом.

axipy.gui.**view_service**
Экземпляр менеджера содержимого окон.

Type *ViewService*

axipy.gui.**selection_service**
Экземпляр доступа к выделенным объектам.

Type *SelectionService*

12.1.7.1 Инструмент окна карты - MapTool

class axipy.gui.**MapTool**
Инструмент окна карты.

При создании своего инструмента новый инструмент наследуется от этого класса, и переопределяет необходимые обработчики событий.

PassEvent
Передать событие дальше. Значение `False`.

Type bool

BlockEvent
Прекратить обработку события. Значение `True`.

Type bool

Пример:

```
MyTool(MapTool):

    def mousePressEvent(self, event):
        print('mouse pressed')
        return self.PassEvent
```

keyPressEvent(*event*)
Обрабатывает событие нажатия клавиши клавиатуры.

Параметры event (QKeyEvent) – Событие нажатия клавиши клавиатуры.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

keyReleaseEvent(*event*)
Обрабатывает событие отпускания клавиши клавиатуры.

Параметры event (QKeyEvent) – Событие отпускания клавиши клавиатуры.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

mouseDoubleClickEvent(*event*)
Обрабатывает событие двойного клика мыши.

Параметры **event** (QMouseEvent) – Событие двойного клика мыши.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

mouseMoveEvent(*event*)
Обрабатывает событие перемещения мыши.

Параметры **event** (QMouseEvent) – Событие перемещения мыши.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

mousePressEvent(*event*)
Обрабатывает событие нажатия клавиши мыши.

Параметры **event** (QMouseEvent) – Событие нажатия клавиши мыши.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

mouseReleaseEvent(*event*)
Обрабатывает событие отпускания клавиши мыши.

Параметры **event** (QMouseEvent) – Событие отпускания клавиши мыши.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

paintEvent(*event*, *painter*)
Обрабатывает событие отрисовки.

Параметры

- **event** (QPaintEvent) – Событие отрисовки.
- **painter** (QPainter) – QPainter для рисования поверх виджета

Тип результата None

to_scene(*device*)
Переводит точки из координат окна(пикселей) в координаты на карте.

Параметры **device** (Union[*Pnt*, *Rect*, QRectF, QPointF, QLineF, QPolygonF]) – Точки в координатах окна.

Тип результата Union[*Pnt*, *Rect*, QRectF, QPointF, QLineF, QPolygonF]

Результат Точки в координатах карты.

property view
Отображение данных в окне.

Тип результата *View*

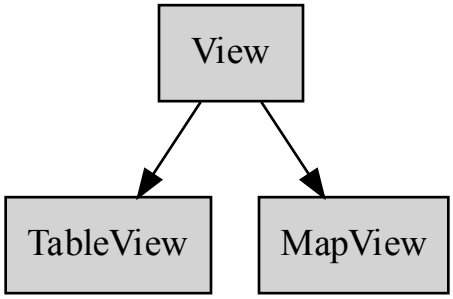
wheelEvent(*event*)
Обрабатывает событие колеса мыши.

Параметры **event** (QWheelEvent) – Событие колеса мыши.

Тип результата bool

Результат *PassEvent*, чтобы пропустить событие дальше по цепочке обработчиков. Или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

12.1.7.2 Базовый класс для отображения данных в окне - View



class axipy.gui.**View**
Базовый класс для отображения данных в окне.

property title
Заголовок окна просмотра.

Тип результата str

property widget
Виджет, соответствующий содержимому окна.

Тип результата QWidget

Результат Qt5 виджет содержимого.

12.1.7.3 Таблица просмотра атрибутивной информации - TableView

class axipy.gui.**TableView**

Базовые классы: *axipy.gui.View*

Таблица просмотра атрибутивной информации. Для создание экземпляра необходимо использовать *axipy.gui.ViewService.create_tableview()* через экземпляр *view_service*

property data_object

Таблица, на основании которой создается данное окно просмотра.

Тип результата *DataObject*

Результат Таблица.

12.1.7.4 Окно просмотра карты - MapView

class axipy.gui.**MapView**

Базовые классы: *axipy.gui.View*

Окно просмотра карты. Используется для проведения различных манипуляций с картой. Для создание экземпляра необходимо использовать *axipy.gui.ViewService.create_mapview()* через экземпляр *view_service* (пример см. ниже).

Примечание: При создании „MapView“ посредством *axipy.gui.ViewService.create_mapview()* производится клонирование экземпляра карты *axipy.render.Map* и для последующей работы при доступе к данному объекту необходимо использовать свойство *map*.

Свойство *device_rect* определяет размер самого окна карты, а свойство *scene_rect* - прямоугольную область, которая умещается в этом окне в СК карты.

Преобразование между этими двумя прямоугольниками производится с помощью матриц трансформации *scene_to_device_transform* и *device_to_scene_transform*.

До параметров самой карты можно доступиться через свойство *map*.

Рассмотрим пример создания карты с последующим помещением ее в окно ее просмотра. Далее, попробуем преобразовать объект типа полигон из координат окна экрана в координаты СК слоя посредством *axipy.da.Geometry.affine_transform()*.

```
# Откроем таблицу, создадим на ее базе слой и добавим в карту
table_world = io.openfile('world.tab')
world = Layer.create(table_world)
map = Map([ world ])
# Для полученной карты создадим окно просмотра
mapview = view_service.create_mapview(map)
# Выведем полученные параметры отображения
print('Прямоугольник экрана:', mapview.device_rect)
print('Прямоугольник карты:', mapview.scene_rect)

>>> Прямоугольник экрана: (0.0 0.0) (300.0 200.0)
>>> Прямоугольник карты: (-16194966.287183324 -8621185.324024437) (16789976.
↪633236416 8326222.646170927)
```

```
#Создадим геометрический объект полигон в координатах экрана и преобразуем его в
↪СК карты
poly_device = Polygon([(100,100), (100,150), (150, 150), (150,100)])
# Используя матрицу трансформации, преобразуем его в координаты карты.
poly_scene = poly_device.affine_transform(mapview.device_to_scene_transform)
# Для контроля выведем полученный полигон в виде WKT
print('WKT:', poly_scene.wkt)

>>> WKT: POLYGON ((-5199985.31371008 -147481.338926755, -5199985.31371008 -
↪4384333.3314756, 297505.173026545 -4384333.3314756, 297505.173026545 -147481.
↪338926755, -5199985.31371008 -147481.338926755))
```

property coordsystem
Система координат карты.

Тип результата *CoordSystem*

property coordsystem_changed
Сигнал о том, что система координат изменилась.

Пример:

```
layer_world = Layer.create(table_world)
map = Map([ layer_world ])
mapview = view_service.create_mapview(map)
mapview.coordsystem_changed.connect(lambda: print('CS was changed'))
csLL = CoordSystem.from_prj("1, 104")
mapview.coordsystem = csLL

>>> CS was changed
```

Тип результата Signal

property device_rect
Видимая область в координатах окна (пиксели).

Тип результата *Rect*

Результат Прямоугольник в координатах окна.

property device_to_scene_transform
Объект трансформации из координат окна в координаты карты.

Тип результата QTransform

Результат Объект трансформации.

property editable_layer
Редактируемый слой на карте.

Тип результата Optional[*VectorLayer*]

Результат Редактируемый слой. Если не определен, возвращает None.

property editable_layer_changed
Сигнал о том, что редактируемый слой сменился.

Тип результата Signal

property map
Объект карты.

Тип результата *Map*

Результат Карта.

property scene_rect
Видимая область в координатах карты (в единицах измерения СК).

Тип результата *Rect*

Результат Прямоугольник в координатах карты.

property scene_to_device_transform
Объект трансформации из координат карты в координаты окна.

Тип результата QTransform

Результат Объект трансформации.

12.1.7.5 Доступ к выделенным объектам - SelectionService

class axipy.gui.**SelectionService**
Класс доступа к выделенным объектам.

Примечание: Получить экземпляр сервиса можно в атрибуте *axipy.gui.selection_service*.

---

add(*table*, *ids*)
Добавляет выборку записи таблицы по их идентификаторам.

Параметры

- **table** (*Table*) – Таблица.
- **ids** (Union[List[int], int]) – Идентификаторы записей.

property changed
Signal[] Выделение было изменено.

Тип результата Signal

clear()
Очищает выборку.

property count
Размер выделения, то есть количество выделенных записей (количество элементов в списке идентификаторов).

Тип результата int

get_as_cursor()
Возвращает выборку в виде итератора по записям.

Пример:

```
for f in selection_service.get_as_cursor():
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

Тип результата Iterator[*Feature*]

get_as_table()
Возвращает выборку в виде таблицы. Содержимое таблицы основывается на текущей выборке на момент вызова данного метода. При последующем изменении или сбросе выборки контент данной таблицы не меняется. Результирующей таблице присваивается наименование в формате *data**, которое в последствии можно изменить. При закрытии базовой таблицы данная таблицы так-же закрывается.

Пример:

```
# Получаем таблицу из выборки.
tbl = selection_service.get_as_table()
# Задаем желаемое имя таблицы (необязательно)
tbl.name = 'my_table'
# Регистрация в каталоге (необязательно)
app.mainwindow.catalog().add(tbl)
for f in tbl.items():
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

Тип результата *Table*

property ids
Список идентификаторов выделенных записей.

Тип результата List[int]

remove(*tbl*, *ids*)
Удаляет из выборки записи таблицы по их идентификаторам.

Параметры

- **tbl** (*Table*) – Таблица.
- **ids** (Union[List[int], int]) – Идентификаторы записей.

set(*table*, *ids*)
Создать выборку из записей таблицы по их идентификаторам.

Параметры

- **table** (*Table*) – Таблица.
- **ids** (Union[List[int], int]) – Идентификаторы записей.

property table
Таблицу, являющаяся источником текущего выделения.

Тип результата *Table*

12.1.7.6 Менеджер содержимого окон - ViewService

class axipy.gui.**ViewService**
Менеджер содержимого окон.

Примечание: Используйте готовый экземпляр этого класса *view_service*.

Список 5: Пример использования

```
table = io.openfile(filepath)
m = Map([table])
view_service.create_mapview(m)
```

activate(*view*)
Делает заданное окно активным.

Параметры **view** (*View*) – Содержимое окна.

property active
Текущее активное окно.

Тип результата Optional[*View*]

Результат None, если нет активных окон.

property active_changed
Signal[] Активное окно изменилось.

Тип результата Signal

close(*view*)
Закрывает окно.

Параметры **view** (*View*) – Содержимое окна.

property count
Количество окон.

Тип результата int

property count_changed
Signal[] Количество окно изменилось.

Тип результата Signal

create_mapview(*map*)
Создает окно из для переданного объекта карты.

Параметры **map** (*Map*) – Карта.

Примечание: Переданная карта копируется.

Тип результата *MapView*

Результат Окно карты.

create_tableview(*table*)
Создает окно в виде табличного представления из объекта данных.

Параметры **table** (*Table*) – Таблица.

Тип результата *TableView*

Результат Окно таблицы.

property views
Список всех известных окон.

Тип результата List[*View*]

12.1.7.7 Диалог выбора СК - ChooseCoordSystemDialog

class axipy.gui.**ChooseCoordSystemDialog**(*coordsystem*)
Диалог выбора координатной системы.

chosenCoordSystem()
Возвращает выбранную в диалоге систему координат.

Тип результата *CoordSystem*

12.1.7.8 Кнопка выбора стиля - StyledButton

class axipy.gui.**StyledButton**(*style*, *parent=None*)
Кнопка, отображающая стиль и позволяющая его менять

Примечание: Сигнал *styleChanged* испускается при смене стиля.

Пример добавления кнопки на диалог:

```
style = Style.from_mapinfo("Pen (1, 2, 8421504)  Brush (2, 255, 0)")

class Dialog(QDialog):
    def __init__(self, parent = None):
        QDialog.__init__(self, parent)
        self.pb = StyledButton( style, self)
        self.pb.styleChanged.connect(self.styleResult)
        self.pb.setGeometry(100, 100, 100, 50)

    def styleResult(self):
        print('Стиль изменен', self.pb.style())

dialog = Dialog()
dialog.exec()
```

style()
Результирующий стиль.

Тип результата *Style*

12.1.7.9 Рабочее пространство - Workspace

class axipy.gui.**Workspace**(*catalog*)
Рабочее пространство для сохранения текущего состояния.

Примечание: Данный класс следует использовать в случае, когда отсутствует главное окно приложения *axipy.app.MainWindow* для сохранения или чтения текущего состояния. Если же главное окно присутствует, то можно воспользоваться методами *axipy.app.MainWindow.load_workspace()* для чтения и *axipy.app.MainWindow.save_workspace()* для записи рабочего пространства.

Рабочее пространство можно как сохранять в файл так и читать из него. При чтении из файла рабочего пространства посредством метода *load_file()* все источники данных (таблицы) открываются и добавляются в переданный в конструктор каталог

axipy.da.DataCatalog, а окна с наполнением добавляются в менеджер окон *axipy.gui.ViewService*, доступный через переменную *view_service*.

В случае записи текущего состояния в файл рабочего пространства последовательность обратная рассмотренной. Состояние каталога и менеджера окон записывается в рабочее пространство посредством метода *save_file()*.

Параметры **catalog** (*DataCatalog*) – Каталог с данными.

Пример чтения данных:

```
catalog = DataCatalog()
print('Before: tables({}), views({})'.format(len(catalog), len(view_service)))
ws = Workspace(catalog)
ws.load_file('ws.mws')
print('After: tables({}), views({})'.format(len(catalog), len(view_service)))

>>> Before: tables(0), views(0)
>>> After: tables(5), views(3)
```

Пример записи рабочего пространства:

```
catalog = DataCatalog()
...
ws = Workspace(catalog)
ws.save_file('ws_out.mws')
```

load_file(*fileName*)
Читает из файла рабочего пространства и заносит данные в текущее окружение.

Параметры **fileName** (str) – Наименование входного файла.

save_file(*fileName*)
Сохраняет текущее состояние в файл рабочего пространства.

Параметры **fileName** (str) – Наименование выходного файла.

12.1.8 axipy.menubar

Модуль меню главного окна Аксиомы.ГИС.

class axipy.menubar.**Position**(*tab*, *group*)
Положение кнопки в меню.

Параметры

- **tab** (str) – Название вкладки.
- **group** (str) – Название группы.

add(*button*, *size=2*)
Добавляет кнопку текущее положение.

Параметры

- **button** (Union[QAction, *ToolDefinition*]) – Кнопка.
- **size** (int) – Размер кнопки. Маленькая кнопка - 1. Большая кнопка - 2.

class axipy.menubar.**ToolDefinition**(*action*, *factory*)
Кнопка с инструментом для добавления в меню.

axipy.menubar.**create_button**(*title*, *on_click*, *icon=''*)
Создает кнопку с заданными параметрами.

Параметры

- **title** (str) – Текст.
- **on_click** (Optional[Callable]) – Действие на нажатие. Любой функтор без параметров.
- **icon** (Union[str, QIcon]) – Иконка. Может быть путем к файлу или адресом ресурса.

Тип результата QAction

Результат Кнопка.

axipy.menubar.**create_tool**(*title*, *on_click*, *icon=''*)
Создает инструмент с заданными параметрами.

Параметры

- **title** (str) – Текст.
- **on_click** (Callable[[], MapTool]) – Фабрика инструмента. Любой функтор без параметров, возвращающий новый инструмент. Чаще всего само объявление инструмента.
- **icon** (Union[str, QIcon]) – Иконка. Может быть путем к файлу или адресом ресурса.

Тип результата *ToolDefinition*

Результат Кнопка с инструментом.

Пример:

```
tool = create_tool("Мой инструмент", on_click=MyTool)
```

axipy.menubar.**get_position**(*tab*, *group*)
Возвращает положение в меню. Может заранее не существовать.

Параметры

- **tab** (str) – Название вкладки.
- **group** (str) – Название группы.

Результат Положение для кнопки.

Пример:

```
pos = get_position("Основные", "Команды")
```

axipy.menubar.**remove**(*action*)
Удаляет кнопку из меню.

Параметры **action** (Union[QAction, *ToolDefinition*]) – Удаляемая кнопка.

12.1.9 axipy.render

Модуль отрисовки.

Данный модуль содержит инструменты, предназначенные для отрисовки геопространственных и прочих данных.

Карта

12.1.9.1 Карта - Map

class axipy.render.**Map**(*layers=[]*)

Класс карты. Рассматривается как группа слоев, объединенная в единую сущность. Вне зависимости от СК входящих в карту слоев, карта отображает все слои в одной СК. Найти наиболее подходящую для этого можно с помощью *get_best_coordsystem()* или же установить другую.

Единицы измерения координат так же берутся из наиболее подходяще СК, но при желании они могут быть изменены. К примеру, вместо метров могут быть установлены километры. Единицы измерения расстояний *distanceUnit* и площадей *areaUnit* берутся из настроек по умолчанию.

Параметры layers (List[*Layer*]) – Список слоев, с которым будет создана карта.

Исключение ValueError – Если один и тот же слой был передан несколько раз.

Пример:

```
table_world = io.openfile('world.tab')
world = Layer.create(table_world)
map = Map([ world ])
print('СК:', map.get_best_coordsystem().prj)
print('Охват:', map.get_best_rect())
print('Единицы измерения расстояний:', map.distanceUnit.description)
map.distanceUnit = unit.mi
print('Единицы измерения расстояний (изменено):', map.distanceUnit.description)

>>> СК: Earth Projection 12, 62, "m", 0
>>> Охват: (-16194966.287183324 -8621185.324024437) (16789976.633236416 8326222.
↪646170927)
>>> Единицы измерения расстояний: километры
>>> Единицы измерения расстояний (изменено): мили
```

property areaUnit

Единицы измерения площадей карты.

Тип результата *AreaUnit*

property distanceUnit

Единицы измерения расстояний на карте.

Тип результата *LinearUnit*

draw(*context*)

Рисует карту в контексте.

Параметры context (*Context*) – Контекст рисования.

Пример:

```
# Пример получения карты как растра
map = Map([ worldcap, world ])
image = QImage(1600, 800, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
map.draw(context)
```

property editable_layer

Слой, установленный для текущего редактирования в карте.

Исключение ValueError – При попытке установить слой, не принадлежащий этой карте.

Тип результата *VectorLayer*

get_best_coordsystem()

Определяет координатную системы карты, наиболее подходящую исходя из содержимого перечня слоев.

Тип результата *CoordSystem*

get_best_rect(*coordsystem=None*)

Определяет ограничивающий прямоугольник карты.

Параметры coordsystem (Optional[*CoordSystem*]) – Координатная система, в которой необходимо получить результат. Если отсутствует, будет выдан результат для наиболее подходящей координатной системы.

Тип результата *Rect*

property layers

Список слоев.

Примеры доступа:

```
len(map.layers)
>>> 2

map.layers[1].name
>>> world

for l in map.layers:
    print('Слой:', l.name)
>>> Слой: world
```

Тип результата *ListLayers*

property need_redraw

Signal[] Сигнал о необходимости перерисовки карты. Возникает при изменении контента одного или нескольких слоев карты. Это может быть обусловлено изменением данных таблиц.

Пример:

```
# Смотрим активное окно.
if isinstance(view_service.active, MapView):
    # Если это карта, подключимся к событию обновления окна этой карты.
    map_view = view_service.active
    map_view.map.need_redraw.connect(lambda : print('Update map'))
```

Тип результата Signal

to_image(*width*, *height*, *coordsystem=None*, *bbox=None*)
Рисует карту в изображение.

Параметры

- **width** (int) – Ширина выходного изображения.
- **height** (int) – Высота выходного изображения.
- **coordsystem** (Optional[*CoordSystem*]) – Координатная система. Если не задана, берется наиболее подходящая.
- **bbox** (Optional[*Rect*]) – Ограничивающий прямоугольник. Если не задан, берется у карты.

Тип результата QImage

Результат Изображение.

property unit
Единицы измерения координат карты.

Тип результата *LinearUnit*

Список слоев карты - ListLayers

class axipy.render.**ListLayers**
Перечень слоев карты.

add(*layer*)
Добавляет слой в карту.

Параметры **layer** (*Layer*) – Добавляемый слой.

Исключение **ValueError** – Если слой уже содержится в карте.

at(*idx*)
Возвращает слой по его индексу. Так же допустимо получение слоя по индексу.

Параметры **idx** (int) – Индекс слоя в списке.

Например:

```
layers.at(2)
layers[2]
```

Тип результата *Layer*

property count
Количество слоев. Так же допустимо использование функции len()

Тип результата int

move(*from_idx*, *to_idx*)
Перемещает слой в списке слоев по его индексу.

Параметры

- **from_idx** (int) – Индекс слоя для перемещения.
- **to_idx** (int) – Целевой индекс.

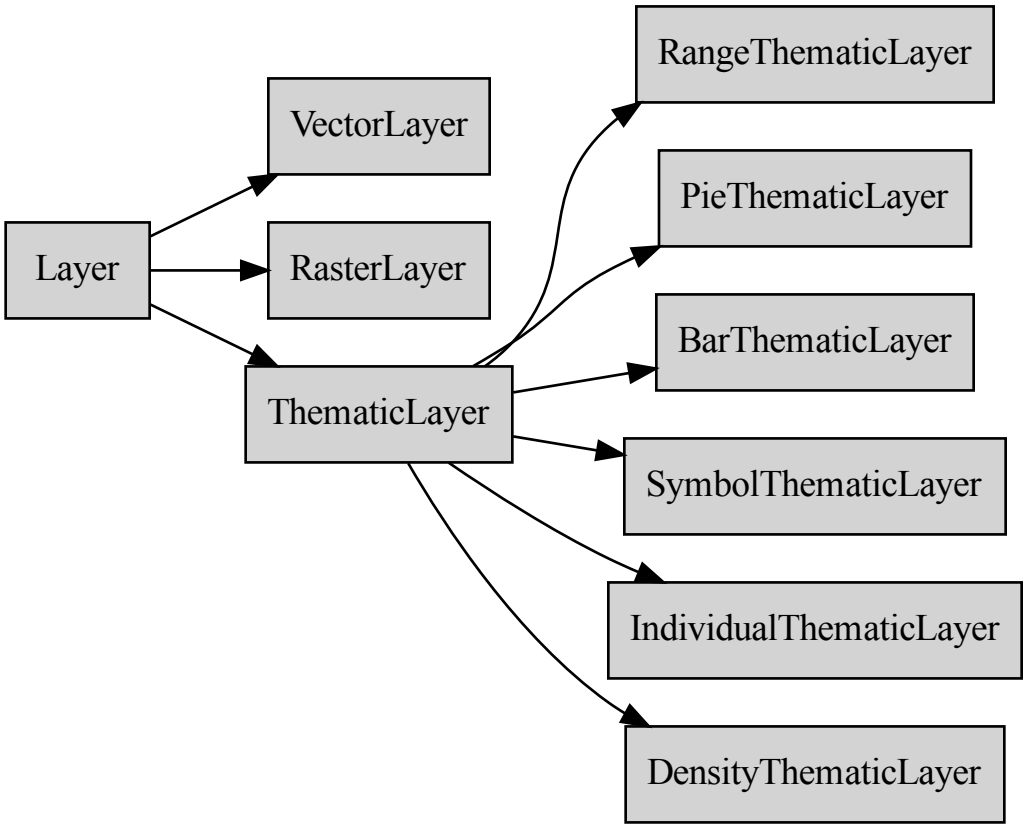
remove(*idx*)
Удаляет слой по индексу.

Параметры **idx** (int) – Индекс удаляемого слоя.

Слои карт

12.1.9.2 Слой - Layer

Иерархия классов слоев карты:



class axipy.render.**Layer**
Абстрактный базовый класс для слоя карты.

Для создания нового экземпляра для векторного или растрового источника данных необходимо использовать метод *create()*. Для тематических слоев - использовать соответствующие им конструкторы.

property bounds
Область в которую попадают все данные, которые могут быть отображены на слое.

Тип результата *Rect*

property coordsystem
Координатная система, в которой находятся данные, отображаемые слоем.

Тип результата *CoordSystem*

classmethod create(*dataObject*)
Создает слой на базе открытой таблицы или растра.

Параметры dataObject (*DataObject*) – Таблица или растр. В зависимости от переданного объекта будет создан *VectorLayer* или *RasterLayer*.

Пример:

```
# Пример создания слоя на базе таблицы
table = io.openfile('world.tab')
world = Layer.create(table)
```

Тип результата *Layer*

property data_changed
Signal[] Сигнал об изменении контента слоя.

Тип результата Signal

property data_object
Источник данных для слоя.

Тип результата *DataObject*

property need_redraw
Signal[] Сигнал о необходимости перерисовать слой.

Пример:

```
# Подпишемся на обновление контента слоя
layer.need_redraw.connect(lambda: print('Update layer'))
```

Тип результата Signal

property opacity
Прозрачность слоя в составе карты. Доступные значения от 0 до 100.

Тип результата int

property title
Наименование слоя.

Тип результата str

Векторный слой - VectorLayer

class axipy.render.**VectorLayer**

Базовые классы: *axipy.render.Layer*

Слой, основанный на базе векторных данных.

Примечание: Создание слоя производится посредством метода вызова *Layer.create()*

---

property label
Метки слоя.

Зададим в качестве формулы метки атрибут «Страна» и запретим перекрытие меток друг другом:

```
world.label.text = "Страна"
world.label.placementPolicy = Label.DISALLOW_OVERLAP
```

Тип результата *Label*

property linesDirectionVisibile
Показ направлений линий.

Тип результата bool

property nodesVisible
Показ узлов линий и полигонов.

Тип результата bool

property overrideStyle
Переопределяемый стиль слоя. Если задан как None (по умолчанию), объекты будут отображены на основании оформления источника данных.

Пример:

```
style_lay = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255) Symbol (33,255,
↪14)")
world.overrideStyle = style_lay
```

Для сброса переопределения достаточно задать значение None:

```
world.overrideStyle = None
```

Тип результата *Style*

property showCentroid
Показ центроидов на слое.

Тип результата bool

property thematic
Перечень тематик для данного слоя.

Тип результата *ListThematic*

Перечень тематик для векторного слоя - ListThematic

class axipy.render.**ListThematic**

Список тематических слоев (тематик) карты.

add(*lay*)

Добавить тематику.

Параметры **lay** (*ThematicLayer*) – Добавляемый тематический слой.

at(*idx*)

Получение тематики по ее индексу.

Параметры **idx** (int) – Индекс запрашиваемой тематики.

Тип результата *ThematicLayer*

property count

Количество тематик слоя.

Тип результата int

move(*fromIdx, toIdx*)

Поменять тематики местами.

Параметры

- **fromIdx** (int) – Текущий индекс.
- **toIdx** (int) – Новое положение.

remove(*idx*)

Удалить тематику.

Параметры **idx** (int) – Индекс удаляемого слоя.

Метка для векторного слоя - Label

class axipy.render.**Label**

Метки слоя. Доступны через свойство векторного слоя label.

property placementPolicy

Принцип наложения меток на слой карты.

Таблица 6: Допустимые значения:

Константа	Значение	Описание
ALLOW_OVERLAP	0	Допускать перекрытие меток (по умолчанию)
DISALLOW_OVERLAP	1	Не допускать перекрытие меток
TRY_OTHER_POSITION	2	Пробовать найти для метки новую позицию

Тип результата int

property text

Наименование атрибута таблицы либо выражение для метки, которое может основываться на одном или нескольких атрибутах..

Тип результата str

Растровый слой - RasterLayer

class axipy.render.**RasterLayer**

Базовые классы: *axipy.render.Layer*

Класс, который должен использоваться в качестве базового класса для тех слоев, в которых используются свойства отрисовки растрового изображения.

Пример:

```
# Создание слоя производится посредством вызова метода Layer.create
raster = io.openfile('TrueMarble.tif')
rasterLayer = Layer.create(raster)
```

property brightness
Яркость. Значение может быть в пределах от 0 до 100.

Тип результата int

property contrast
Контраст. Значение может быть в пределах от 0 до 100.

Тип результата int

property grayscale
Черно-белое изображение.

Тип результата bool

property transparentColor
Цвет растра, который обрабатывается как прозрачный.

Тип результата QColor

Тематика

12.1.9.3 Тематика - ThematicLayer

class axipy.render.**ThematicLayer**

Базовые классы: *axipy.render.Layer*

Абстрактный класс слоя с тематическим оформлением векторного слоя карты на базе атрибутивной информации.

Интервалы - RangeThematicLayer

class axipy.render.**RangeThematicLayer**(*expression*)

Базовые классы: *axipy.render.ThematicLayer*, *axipy.render.StyledByIndexThematic*

Тематическое оформление слоя с распределением значений по интервалам.

Параметры expression (str) – Наименование атрибута или выражение.

Пример:

```
# Пример создания тематики с последующим добавлением ее к базовому слою
range = RangeThematicLayer("Население")
range.ranges = 6
range.splitType = RangeThematicLayer.EQUAL_COUNT
world.thematic.add(range)
```

assign_two_colors(*colorMin, colorMax*)
Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** (QColor) – Цвет нижнего диапазона.
- **colorMax** (QColor) – Цвет верхнего диапазона.

get_interval_value(*idx*)
Возвращает предельные значения для указанного интервала в виде пары значений.

Параметры **idx** (int) – Индекс диапазона.

Тип результата []

property ranges
Количество интервалов.

Тип результата int

set_interval_value(*idx, v*)
Заменяет предельные значения интервала.

Параметры

- **idx** (int) – Индекс диапазона.
- **v** (*[ ]*) – Значение.

Пример запроса с последующей заменой:

```
v = world.thematic[0].get_interval_value(2) # Запрос
v = (999, v[1]) # Заменим минимальное значение для интремала с индексом 2
world.thematic[0].set_interval_value(2, v) # Замена
```

property splitType
Тип распределения значений по интервалам.

Допустимые значения:

EQUAL_INTERVAL: Распределение исходя из равномерности интервалов (по умолчанию).

EQUAL_COUNT: Распределение исходя их равного количества объектов в каждом интервале.

MANUAL: Ручное распределение значений путем задания пределов вручную.

Тип результата int

Круговые диаграммы - PieThematicLayer

class axipy.render.**PieThematicLayer**(*expressions*)

Базовые классы: *axipy.render.ThematicLayer*, *axipy.render.AllocationThematic*, *axipy.render.OrientationThematic*, *axipy.render.StyledByIndexThematic*

Тематика в виде круговых диаграмм.

Параметры expressions (List) – Наименования атрибутов или выражений в виде списка list.

Пример:

```
# Создание тематики с последующим добавлением ее к базовому слою
pie = PieThematicLayer(["Население", "Мужское", "Женское"])
world.thematic.add(pie)
```

property startAngle

Начальный угол отсчета диаграммы.

Тип результата bool

Столбчатые диаграммы - BarThematicLayer

class axipy.render.**BarThematicLayer**(*expressions*)

Базовые классы: *axipy.render.ThematicLayer*, *axipy.render.AllocationThematic*, *axipy.render.OrientationThematic*, *axipy.render.StyledByIndexThematic*

Тематика в виде столбчатых диаграмм.

Параметры expressions (List) – Наименования атрибутов или выражений в виде списка list.

Пример:

```
# Создание тематики с последующим добавлением ее к базовому слою
bar = BarThematicLayer(["Население", "Мужское", "Женское"])
world.thematic.add(bar)
```

property isStacked

Расположение столбчатой диаграммы в виде стопки, если True.

Тип результата bool

Знаки - SymbolThematicLayer

class axipy.render.**SymbolThematicLayer**(*expression*)

Базовые классы: *axipy.render.ThematicLayer*

Тематический слой с распределением по интервалам и с градуировкой символа по размеру.

Параметры expression (str) – Наименование атрибута или выражение.

Пример:

```
# Создание тематики с последующим добавлением ее к базовому слою
symbol = SymbolThematicLayer("Население")
symbol.defaultStyle = Style.from_mapinfo("Symbol (33, 255,14)")
symbol.maxHeight = 34
world.thematic.add(symbol)
```

property defaultStyle
Стиль по умолчанию для оформления знаков.

Тип результата *Style*

property maxHeight
Максимальная высота символа.

Тип результата float

property minHeight
Минимальная высота символа.

Тип результата float

Индивидуальные значения - IndividualThematicLayer

class axipy.render.**IndividualThematicLayer**(*expression*)

Базовые классы: *axipy.render.ThematicLayer*, *axipy.render.StyledByIndexThematic*

Тематический слой с распределением стилей по индивидуальным значением.

Параметры expression (str) – Наименование атрибута или выражение.

Пример:

```
# Создание тематики с последующим добавлением ее к базовому слою
individual = IndividualThematicLayer("Население")
world.thematic.add(individual)
```

property count
Количество значений в тематике.

get_value(*idx*)
Выражение по указанному индексу.

Параметры idx (int) – Индекс.

Тип результата Any

Плотность точек - DensityThematicLayer

class axipy.render.**DensityThematicLayer**(*expression*)

Базовые классы: *axipy.render.ThematicLayer*

Тематический слой с заполнением полигональных объектов точками, плотность которых зависит от вычисленного значения по выражению.

Параметры expression (str) – Наименование атрибута или выражение.

Пример:

```
# Создание тематики с последующим добавлением ее к базовому слою
density = DensityThematicLayer("Население")
world.thematic.add(density)
```

property color
Цвет точек.

Тип результата QColor

property pointForMaximum
Количество точек для максимального значения.

Тип результата int

property size
Размер точек.

Тип результата float

Метод распределения значений для диаграмм - AllocationThematic

class axipy.render.**AllocationThematic**
Метод распределения значений для диаграмм.

property allocationType
Тип распределения значений.

Таблица 7: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

Тип результата int

Ориентация для диаграмм - OrientationThematic

class axipy.render.**OrientationThematic**
Ориентация тематического представления относительно центроида объекта.

property orientationType
Ориентация относительно центроида.

Таблица 8: Допустимые значения.

Константа	Значение	Описание
CENTER	0	Диаграмма рисуется по центру (по умолчанию)
LEFT_UP	1	Диаграмма выравнивается по левому верхнему краю
UP	2	Диаграмма выравнивается по верхнему краю
RIGHT_UP	3	Диаграмма выравнивается по верхнему правому краю
RIGHT	4	Диаграмма выравнивается по правому краю
RIGHT_DOWN	5	Диаграмма выравнивается по нижнему правому краю
DOWN	6	Диаграмма выравнивается по нижнему краю
LEFT_DOWN	7	Диаграмма выравнивается по нижнему левому краю
LEFT	8	Диаграмма выравнивается по левому краю

Тип результата int

Стиль заливки - StyledByIndexThematic

class axipy.render.**StyledByIndexThematic**
Поддержка набора индексированных стилей.

get_style(*idx*)
Стиль для указанного выражения.

Параметры **idx** (int) – Порядковый номер выражения.

Тип результата *Style*

set_style(*idx, style*)
Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** (int) – Индекс.
- **style** (*Style*) – Назначаемый стиль.

Пример установки стиля для значения с индексом 2 первого тематического слоя:

```
style_new = Style.from_mapinfo("Brush (2, 255, 0)")
world.thematic[0].set_style(2, style_new)
```

Легенда

12.1.9.4 Легенда слоя - Legend

class axipy.render.**Legend**(*lay*)

Легенда слоя. Позволяет получить информацию об условных обозначениях на слое.

Параметры **lay** (*Layer*) – Слой, для которого создается легенда.

Пример:

```
# Пример создания легенды для слоя world
legend = Legend(world)
legend.position = (100, 10)
```

property caption

Заголовок легенды.

Пример:

```
legendWorld = Legend(world)
legendWorld.caption = 'Легенда для слоя'
legendWorld.styleCaption = Style.from_mapinfo("Font ("Arial", 0, 9, 255)")
```

Тип результата str

draw(*context*)

Рисует легенду в контексте.

Легенду также можно отрисовать совместно с картой в одном контексте (см. *Map.draw()*).

Параметры **context** (*Context*) – Контекст рисования.

Пример:

```
context = Context(painter)
context.rect = Rect(0,0, 1000, 1000)
legend.draw(context)
```

property position

Положение легенды в контексте рисования.

Тип результата *Pnt*

property styleCaption

Стиль заголовка легенды.

Тип результата *Style*

to_image(*width*, *height*)

Возвращает легенду в виде растра.

Параметры

- **width** (int) – Ширина выходного растра.
- **height** (int) – Высота выходного растра.

Тип результата QImage

Классы, относящиеся к работе с отчетами

12.1.9.5 Отчет - Report

class axipy.render.**Report**(*printer*)

План отчета для последующей печати.

draw(*context*)

Выводит отчета в заданном контексте.

Параметры **context** (*Context*) – Контекст, в котором будет отрисован отчет.

Пример:

```
# Пример создания отчета с геометрическим элементом и вывод его в pdf
printer = QPrinter()
printer.setPaperSize(QPrinter.A4)
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName('report.pdf')
painterReport = QPainter(printer)
contextReport = Context(painterReport)

report = Report(printer)
geometryReportItem = GeometryReportItem()
geometryReportItem.geometry = Geometry.from_wkt('POLYGON ((10 10, 10 100, 100␣
↪100, 10 10))')
geometryReportItem.style = Style.from_mapinfo(mapbasic.brush(45, 255, 65535))
report.items.add(geometryReportItem)
report.draw(contextReport)
```

fill_on_pages()

Максимально заполняет страницу(ы) отчета.

property horisontalPages

Количество страниц отчета по горизонтали.

Тип результата int

property items

Элементы отчета.

Тип результата *ReportItems*

property name

Наименование отчета.

Тип результата str

property need_redraw

Signal[] Сигнал о необходимости перерисовки части или всего отчета.

Параметры **rect** (Union[*Rect*, QRectF]) – Часть отчета, которую необходимо обновить.

Тип результата Signal

property pageSize

Размеры одного листа отчета.

Тип результата QSizeF

property unit
Единицы измерения в отчете.

Тип результата *LinearUnit*

property verticalPages
Количество страниц отчета по вертикали.

Тип результата int

Список элементов отчета - ReportItems

class axipy.render.**ReportItems**
Список элементов отчета.

add(*item*)
Добавляет новый элемент в отчет.

Параметры item (*ReportItem*) – Вставляемый элемент

at(*idx*)
Возвращает элемент отчета по его индексу.

Параметры idx (int) – Индекс.

Тип результата *ReportItem*

Результат Элемент отчета. Возвращает None в случае, если не найдено.

property count
Количество элементов отчета в текущем отчете на данный момент.

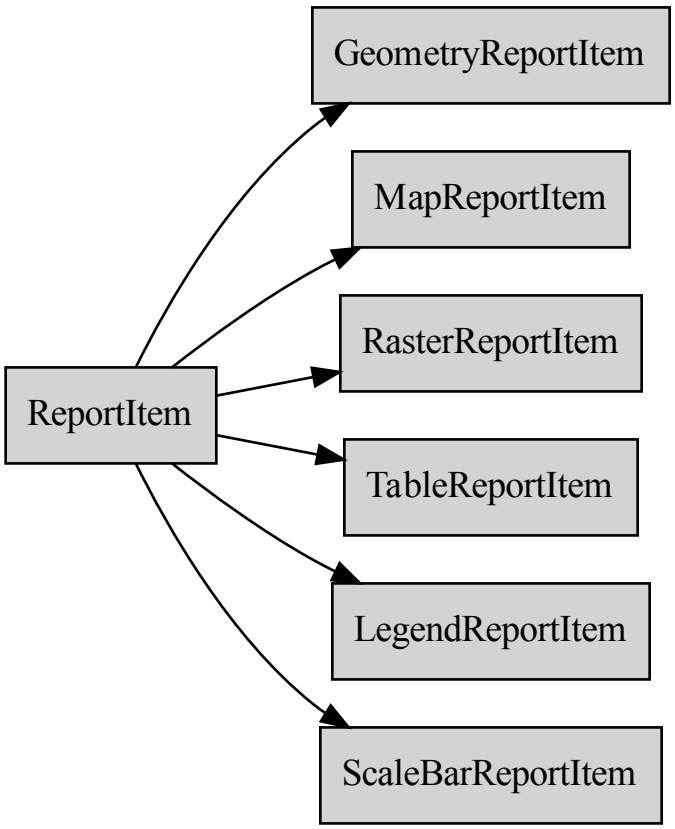
Тип результата int

remove(*idx*)
Удаляет элемент по его индексу. Если индекс корректен, элемент будет удален.

Параметры idx (int) – Индекс удаляемого элемента.

12.1.9.6 Элемент отчета - ReportItem

Элементы отчета:



class axipy.render.**ReportItem**

Базовый класс элемента отчета.

property borderStyle

Стиль обводки элемента отчета.

Тип результата *Style*

property fillStyle

Стиль заливки элемента отчета.

Тип результата *Style*

intersects(*checkRect*)

Пересекается ли с переданным прямоугольником.

Параметры checkRect (Union[*Rect*, QRectF]) – Прямоугольник для анализа.

property rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Тип результата *Rect*

Элемент отчета: геометрия - GeometryReportItem

class axipy.render.**GeometryReportItem**

Базовые классы: *axipy.render.ReportItem*

Элемент отчета типа геометрия.

property geometry
Геометрическое представление объекта.

Тип результата *Geometry*

property style
Стиль геометрического представления объекта.

Тип результата *Style*

Элемент отчета: карта - MapReportItem

class axipy.render.**MapReportItem**(*rect*, *map*)

Базовые классы: *axipy.render.ReportItem*

Элемент отчета, основанный на созданной ранее карте.

Примечание: Перед созданием элемента отчета необходимо предварительно создать карту, на основе которой будет создан элемент отчета.

Параметры

- **rect** (Union[*Rect*, QRectF]) – Размер элемента отчета в единицах измерения отчета.
- **map** (*Map*) – Карта, на базе которой будет создан элемент отчета.

Пример:

```
mapReportItem = MapReportItem(Rect(10, 10, 200, 100), map)
mapReportItem.scale = 200000000
report.items.add(mapReportItem)
```

property center
Центр карты в координатах карты.

Тип результата *Pnt*

map()
Возвращает элемент типа карта, на основании которой создается элемент отчета.

Тип результата *Map*

property scale
Текущее значение масштаба карты.

Тип результата float

Элемент отчета: растр - RasterReportItem

class axipy.render.**RasterReportItem**(*rect*, *data*)

Базовые классы: *axipy.render.ReportItem*

Элемент отчета, основанный на растре.

Примечание: В качестве источника может быть как локальный файл, расположенный в файловой системе, так и базе растра, размещенного на Web ресурсе.

Параметры

- **rect** (Union[*Rect*, QRectF]) – Размер элемента отчета в единицах измерения отчета.
- **data** (str) – Путь к растровому файлу или его URL.

Пример:

```
# Растр на базе URL
rasterReportItem = RasterReportItem(Rect(10, 10, 140, 70),
    'https://upload.wikimedia.org/wikipedia/en/7/72/World_Map_WSF.svg.png')
report.items.add(rasterReportItem)
```

Пример:

```
# Растр на базе локального файла
rasterReportItem = RasterReportItem(Rect(10, 10, 140, 70), 'TrueMarble.tif')
report.items.add(rasterReportItem)
```

property preserveAspectRatio
Сохранять пропорции при изменении размеров элемента.

Тип результата bool

Элемент отчета: таблица - TableReportItem

class axipy.render.**TableReportItem**(*rect*, *table*)

Базовые классы: *axipy.render.ReportItem*

Элемент отчета табличного представления данных.

Примечание: Позволяет отображать как таблицу целиком, так и накладывая дополнительные ограничения при отображении.

Параметры

- **rect** (Union[*Rect*, QRectF]) – Размер элемента отчета в единицах измерения отчета.
- **table** (*Table*) – Таблица.

Пример:

```
table_world = io.openfile('world.tab')
tableReportItem = TableReportItem(Rect(10, 150, 180, 100), table_world)
tableReportItem.columns = table_world.schema.attribute_names()[:3] # Берем для␣
↪показа первые три атрибута
tableReportItem.rowFrom = 5 # С 5-й сторки
tableReportItem.rowCount = 5 # Показываем 5 строк
tableReportItem.startNumber = 5 # Нумерация с 5
tableReportItem.borderStyle = Style.from_mapinfo("Pen (1, 2, 0)") # Стиль рамки
tableReportItem.fillStyle = Style.from_mapinfo("Brush (8, 65535)") # Стиль фона
report.items.add(tableReportItem)
```

property columns
Перечень наименований для отображения. Если задать пустой список, будут отображены все поля таблицы.

Тип результата list

refreshValues()
«Обновление данных из таблицы.

property rowCount
Количество записей. Если указано -1, то берутся все оставшиеся записи.

Тип результата int

property rowFrom
Номер первой строки из таблицы или запроса.

Тип результата int

property startNumber
Нумерация записей. Порядковый номер первой записи.

Тип результата int

table()
Базовая таблица или запрос.

Тип результата *Table*

Элемент отчета: легенда - LegendReportItem

class axipy.render.**LegendReportItem**(*rect*, *legend*)

Базовые классы: *axipy.render.ReportItem*

Элемент отчета, основанный на легенде векторного или тематического слоя.

Параметры

- **rect** (Union[*Rect*, QRectF]) – Размер элемента отчета в единицах измерения отчета.

- **legend** (*Legend*) – Предварительно созданная легенда. Она может относиться как к векторному, так и к тематическому слою.

Пример создания легенды для тематического слоя:

```
world = Layer.create(table_world) # Базовый слой
range_ = RangeThematicLayer("Население") # Тематический слой
world.thematic.add(range_)
legend = Legend(range_) # Легенда
legendReportItem = LegendReportItem(Rect(100, 130, 50, 70), legend) # Элемент␣
↪отчета
report.items.add(legendReportItem)  # Добавляем в отчет
```

Элемент отчета: масштабная линейка - ScaleBarReportItem

class axipy.render.**ScaleBarReportItem**(*rect*, *map*)

Базовые классы: *axipy.render.ReportItem*

Элемент отчета - масштабная линейка для карты.

Пример:

```
scaleBarReportItem = ScaleBarReportItem(Rect(120, 130, 80, 50), mapReportItem)
report.items.add(scaleBarReportItem)
```

12.1.9.7 Контекст рисования - Context

class axipy.render.**Context**(*painter*)

Контекст рисования.

Содержит информацию о том, куда производится рисование (QPainter), а так же о необходимых преобразованиях, которые необходимо применить к объекту непосредственно перед его отрисовкой.

Параметры painter (QPainter) – Объект QPainter для рисования.

Пример создания контекста на базе растра. Далее его можно использовать для отрисовки карты *Map*, отчета *Report* или легенды *Legend*:

```
image = QImage(1600, 800, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
```

property coordsystem

Координатная система.

Если она не задана, берется наиболее подходящая исходя из текущего контента.

Тип результата *CoordSystem*

property dpi

Количество точек на дюйм, с которым происходит рисование.

Влияет на отрисовку в «реальных» единицах измерения (мм, см, пункты).

Тип результата float

property rect
Прямоугольник в координатах карты, который будет отрисован.

Тип результата *Rect*

12.1.10 axipy.utl

12.1.10.1 Точка - Pnt

class axipy.utl.**Pnt**(*x*, *y*)
Точка без геопривязки. Может быть использована в качестве параметра геометрии (точки полигона) или при получении параметров, где результат представлен в виде точки (центр карты или элемента отчета).

Параметры

- **x** (float) – X координата.
- **y** (float) – Y координата.

property x
Координата X.

Тип результата float

property y
Координата Y.

Тип результата float

12.1.10.2 Прямоугольник - Rect

class axipy.utl.**Rect**(*xmin*, *ymin*, *xmax*, *ymax*)
Прямоугольник, который не обладает геопривязкой. Используется для различного вида запросов.

property center
Центр прямоугольника.

Тип результата *Pnt*

contains(*p*)
Содержит ли в своих пределах точку.

Параметры **p** (*Pnt*) – Тестируемая точка.

Тип результата bool

property height
Высота прямоугольника.

property is_empty
Если один или оба размера равны нулю.

property is_valid
Является ли прямоугольник правильным.

normalize()
Исправляет прямоугольник, если его ширина или высота отрицательны.

property width
Ширина прямоугольника.

property xmax
Максимальное значение X.

Тип результата float

property xmin
Минимальное значение X.

Тип результата float

property ymax
Максимальное значение Y.

Тип результата float

property ymin
Минимальное значение Y.

Тип результата float

Содержание модулей Python

a

Алфавитный указатель

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y