
ахіру
Версия 7.1

ООО ЭСТИ

12 февраля, 2026

Оглавление

1	О библиотеке	1
2	Использование в Аксиоме	3
3	Использование среды разработки	9
3.1	PyCharm Windows	10
3.2	PyCharm Linux	14
3.3	VS Code Windows	22
3.4	VS Code Linux	26
4	Системы Координат	31
5	Объекты данных	33
5.1	Таблицы	33
5.2	Растры	39
6	Провайдеры данных	43
6.1	Открытие/Создание	44
6.2	Импорт/Экспорт	47
7	Записи	55
7.1	Атрибуты	55
7.2	Геометрический атрибут	56
7.3	Стиль для геометрического атрибута	56
7.4	Идентификаторы записей	56
8	Геометрия	57
8.1	Типы	57
8.2	Свойства геометрии	64
8.3	Сериализация	64
8.4	Преобразования	64
8.5	Пространственные операции	65
9	Стиль	73
10	Окна просмотра	77
10.1	Слой	77

10.2 Карта	78
10.3 Тематические слои	83
10.4 Легенда	86
10.5 Отчет	89
11 Создание кнопок	91
11.1 Создание кнопки	91
11.2 Доступность кнопки	91
12 Создание виджетов	93
12.1 Программное наполнение диалога	94
12.2 Использование файла ресурсов *.ui. Прямая загрузка.	95
12.3 Использование наследования на базе файла ресурсов *.ui	98
13 Создание инструментов	101
13.1 Передача параметров в инструменты	101
13.2 Панель активного инструмента	101
14 Работа с длительными операциями	103
14.1 Задачи	103
14.2 Представление прогресса операции	105
14.3 Создание пользовательского виджета для отображения прогресса	105
14.4 Выполнение задач и многопоточность	107
15 Разработка Плагинов (Модулей)	109
15.1 Структура плагина	109
15.2 Класс Plugin	113
15.3 Архив	114
15.4 Зависимости	114
15.5 Рекомендации по написанию плагинов	118
16 Создание приложения	119
16.1 Пример приложения из готовых компонентов	123
17 axiру - Основной пакет библиотеки для взаимодействия с ГИС Аксиома.	129
17.1 Инициализация Аксиомы	129
17.2 Plugin - Плагин ГИС Аксиома	130
17.3 PluginManager - Менеджер плагинов	132
17.4 PluginInfo - Информация о модуле	134
17.5 Настройки ГИС Аксиома	134
17.6 Вспомогательные функции	140
17.7 Менеджеры	142
17.8 Диалоги	143
18 axiру.app - Модуль приложения.	147
18.1 MainWindow - Главное окно	147
18.2 Version - Информация о версии	150
19 axiру.cs - Модуль систем координат.	153
19.1 CoordSystem - Система Координат (СК)	153
19.2 CoordTransformer - Трансформация координат	159
19.3 LinearUnits - Единицы измерения расстояний	160
19.4 AreaUnits - Единицы измерения площадей	162
19.5 UnitValue - Значение вместе с единицей измерения	165

20	axipy.concurrent - Модуль для работы с длительными задачами в фоновом потоке.	167
20.1	Task - Пользовательская задача	167
20.2	DialogTask - Пользовательская задача с диалогом	173
20.3	TaskManager - Сервис для отслеживания пользовательских задач	179
21	axipy.utl - Вспомогательные классы.	183
21.1	Pnt - Точка	183
21.2	Rect - Прямоугольник	186
21.3	FloatCoord - Координаты с плавающей точкой.	191
21.4	AngleCoord - Угловые координаты.	193
21.5	CalcMode - Режим расчетов	196
22	axipy.da - Модуль источников данных.	197
22.1	DataProvider - Провайдеры	197
22.2	DataManager - Каталог данных	259
22.3	DataObject - Объект данных	265
22.4	Table - Таблица	267
22.5	QueryTable - SQL запрос.	275
22.6	SelectionTable - Таблица с текущей выборкой.	284
22.7	CosmeticTable - Таблица с данными косметического слоя.	292
22.8	SupportedOperations - Доступные операции	300
22.9	Feature - Запись в таблице	301
22.10	Schema - Схема таблицы	305
22.11	Attribute - Атрибут схемы таблицы	308
22.12	Geometry - Геометрия	313
22.13	Style - Стил	492
22.14	TypeSqlDialect - Диалект при выполнении запросов	531
22.15	Raster - Растр	531
22.16	rastrer - Операции с растром	534
22.17	ObserverManager - Менеджер наблюдателей	536
22.18	Observer - Наблюдатель	539
22.19	TabFile - Файл TAB	540
22.20	RasteredTable - Источники ГИС Панорама и AutoCAD.	541
22.21	GeometryClass - Класс геометрических объектов	543
23	axipy.render - Модуль отрисовки.	545
23.1	Map - Карта	545
23.2	ListLayers - Список слоев карты	550
23.3	Layer - Слой	553
23.4	Legend - Легенда слоя	576
23.5	LegendItem - Элемент легенды	579
23.6	ListLegendItems - Список элементов легенды	579
23.7	ThematicLayer - Тематика	580
23.8	Report - Отчет	616
23.9	Context - Контекст рисования	630
23.10	CustomLabels - Пользовательские метки карты	631
23.11	ClipGeometry - Обрезка карты	632
24	axipy.gui - Модуль пользовательского интерфейса.	635
24.1	ViewTool - Инструмент окна карты или отчета	635
24.2	DeactivationReason - Причина выключения инструмента	641
24.3	ActiveToolPanel - Панель активного инструмента	642
24.4	AxipyActiveToolPanelHandlerBase - Базовый класс обработчика панели активного инструмента	643

24.5	AxipyAcceptableActiveToolHandler - Управление панелью активного инструмента с предустановленными кнопками	645
24.6	AxipyCustomActiveToolPanelHandler - Управление панелью активного инструмента без предустановленных кнопок управления	646
24.7	View - Базовый класс для отображения данных в окне	647
24.8	TableView - Таблица просмотра атрибутивной информации	649
24.9	DrawableView - Базовый класс с поддержкой визуального редактирования геометрий	652
24.10	MapView - Окно просмотра карты	655
24.11	ReportView - Окно просмотра отчета	663
24.12	LegendView - Окно просмотра легенд карты	670
24.13	ListLegend - Список легенд	673
24.14	SelectionManager - Доступ к выделенным объектам	673
24.15	ViewManager - Менеджер содержимого окон	676
24.16	ActionManager - Менеджер системных действий и инструментов	680
24.17	Workspace - Рабочее пространство	682
24.18	ChooseCoordSystemDialog - Диалог выбора СК	684
24.19	PasswordDialog - Диалог аутентификации пользователя.	685
24.20	BoundingRectDialog - Диалог задания параметров прямоугольника	685
24.21	StyleButton - Кнопка выбора стиля	686
24.22	Диалог задания стиля	687
24.23	Виджеты Аксиомы	688
24.24	Notifications - Отправление уведомлений	692
25	axipy.menubar - Модуль меню главного окна ГИС Аксиома.	693
25.1	Button - Кнопка	693
25.2	ActionButton - Кнопка с действием	694
25.3	SystemActionButton - Действие, присутствующее в системе	696
25.4	ToolButton - Кнопка с инструментом	697
25.5	Separator - Разделитель	699
25.6	Position - Положение кнопки	700
26	Глоссарий	701
27	История изменений	703
27.1	7.1 Изменения	703
27.2	7.0 Изменения	703
27.3	6.2 Изменения	705
27.4	6.1 Изменения	705
27.5	6.0 Изменения	706
27.6	5.2 Изменения	707
27.7	5.1 Изменения	707
27.8	5.0.1 Изменения	707
27.9	5.0 Изменения	708
27.10	4.4 Изменения	709
27.11	4.3 Изменения	710
27.12	4.0 Изменения	711
27.13	3.7.0 Изменения	711
27.14	3.5.0 Изменения	711
27.15	3.0.0 Изменения	712
27.16	2.9.0 Изменения	713
28	Лицензия	715
28.1	БЕЗВОЗМЕЗДНЫЙ ЛИЦЕНЗИОННЫЙ ДОГОВОР	716
28.2	ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ, ИСПОЛЬЗУЕМЫЕ В ДОГОВОРЕ	716

28.3 СТАТЬЯ 1. ОБЩИЕ ПОЛОЖЕНИЯ	716
28.4 СТАТЬЯ 2. ПРЕДМЕТ ДОГОВОРА	717
28.5 СТАТЬЯ 3. ДОПОЛНИТЕЛЬНЫЕ УСЛОВИЯ	718
28.6 СТАТЬЯ 4. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ	719
28.7 СТАТЬЯ 5. РЕКВИЗИТЫ ПРАВООБЛАДАТЕЛЯ	720
Содержание модулей Python	721
Алфавитный указатель	723

О библиотеке

Библиотека **axipy** обеспечивает взаимодействие с ГИС Аксиома для решения геоинформационных задач.

axipy использует библиотеку PySide2 (Qt for Python). Библиотека PySide2 имеет лицензию [LGPL](#). Программу, разработанную с использованием библиотеки PySide2, можно лицензировать на свое усмотрение, включая проприетарные и коммерческие лицензии.

Документация к **axipy** состоит из двух частей:

- [руководство разработчика](#);
- [справочник функций](#).

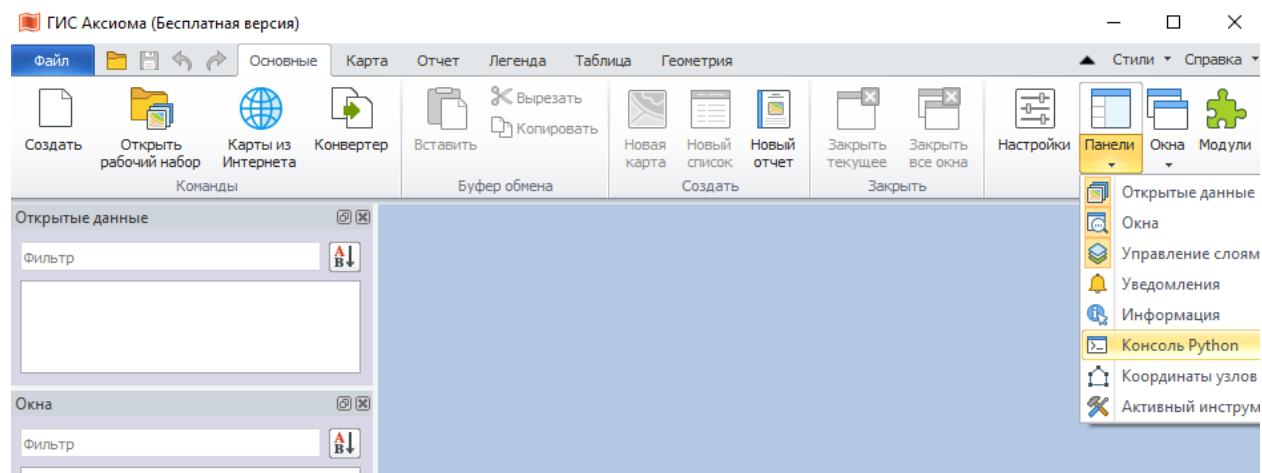
Руководство разработчика логически структурировано. В нем описаны общие принципы решения базовых задач по обработке геопространственных данных. Рекомендуется читать его в прямом порядке от начала до конца, чтобы получить общее представление о возможностях, предоставляемых библиотекой.

Полное описание классов, свойств, методов, исключений, функций и их параметров содержится в справочнике функций.

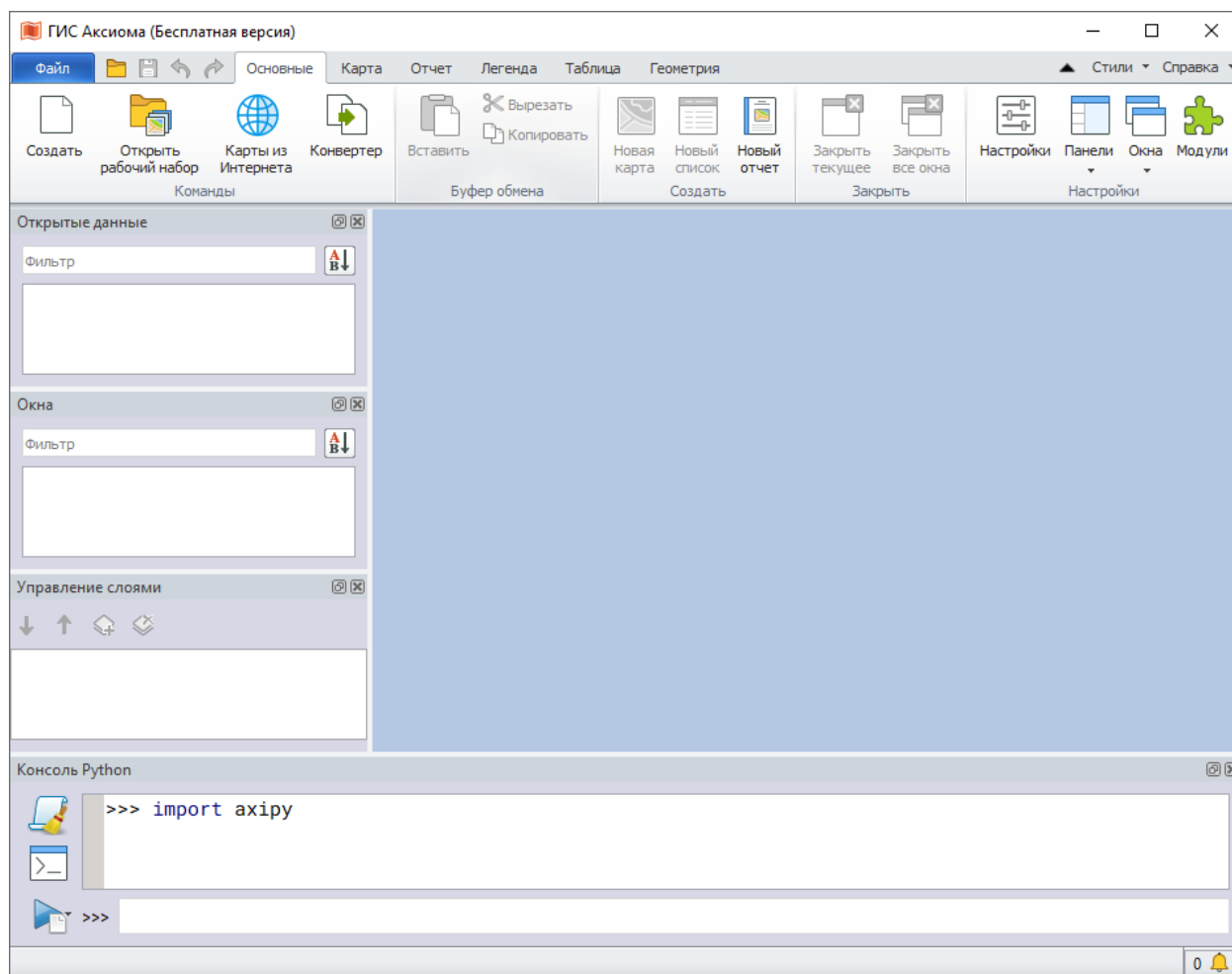
Использование в Аксиоме

В Аксиоме есть встроенные средства для использования `axiру` - консоль Python и редактор кода.

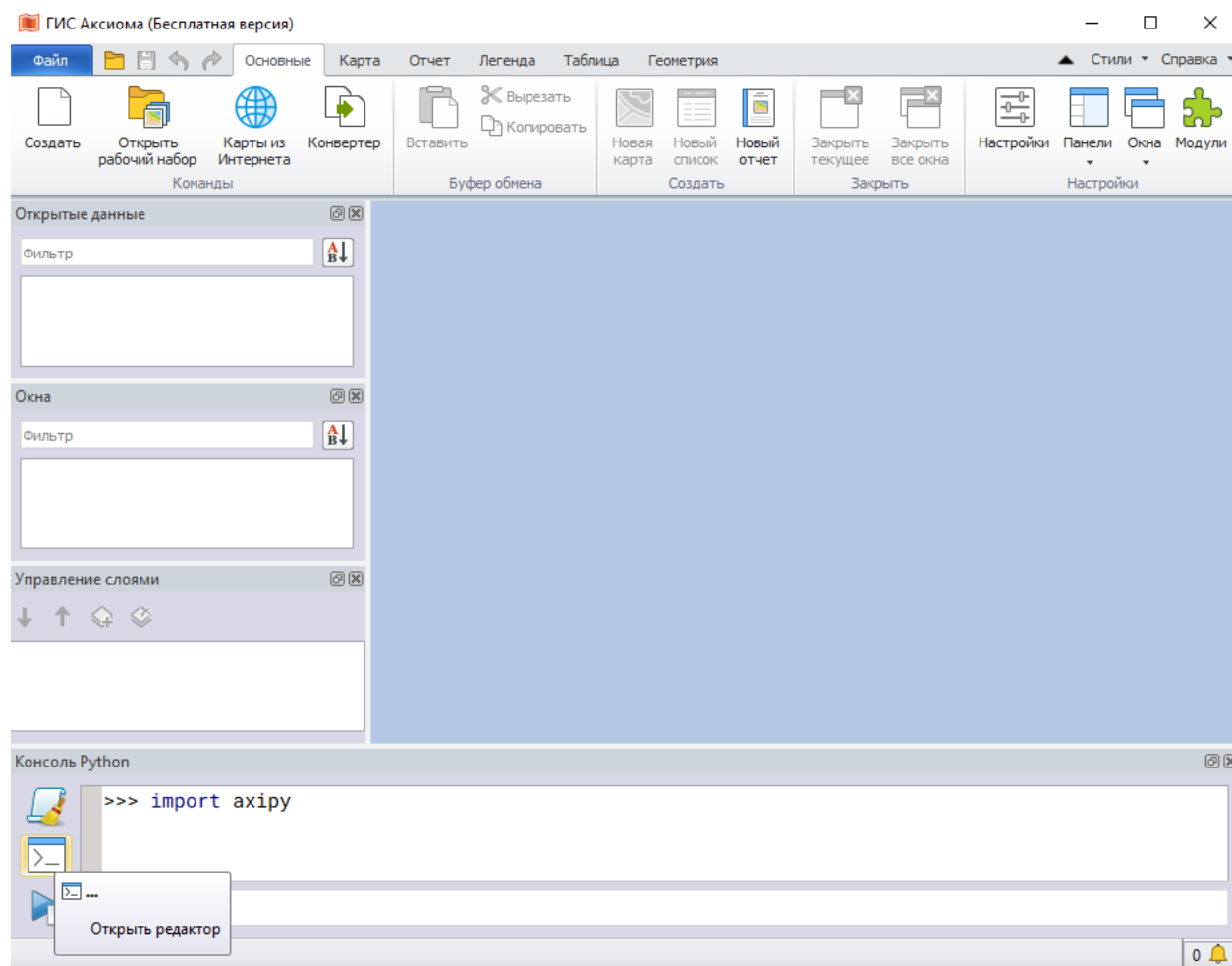
Чтобы открыть консоль Python, на вкладке «Основные» нажмите кнопку «Панели» и в появившемся списке выберите «Консоль Python».



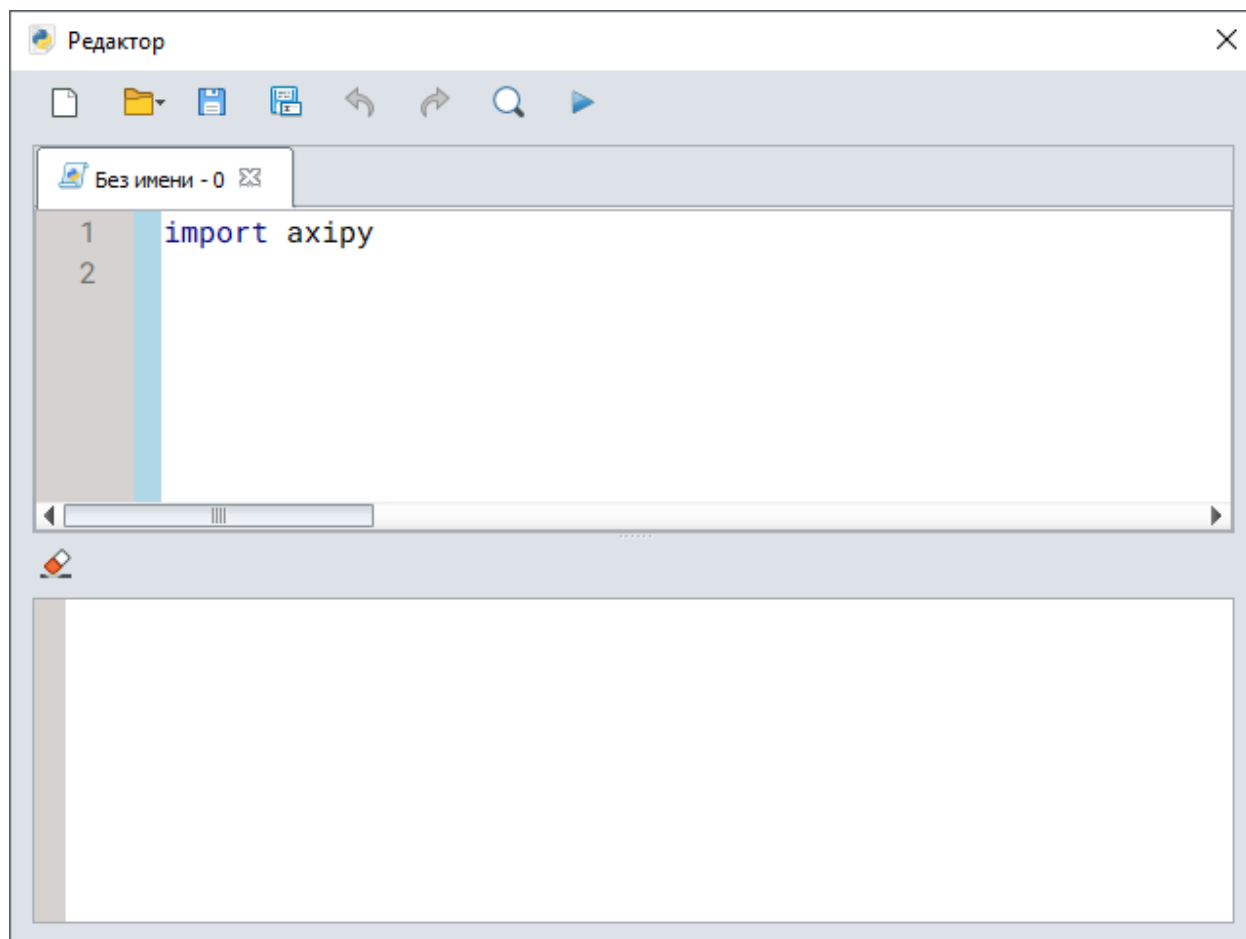
Открытая консоль Python:



Чтобы открыть редактор кода, нажмите кнопку «Открыть редактор» на панели «консоль Python».



Открытый редактор кода:



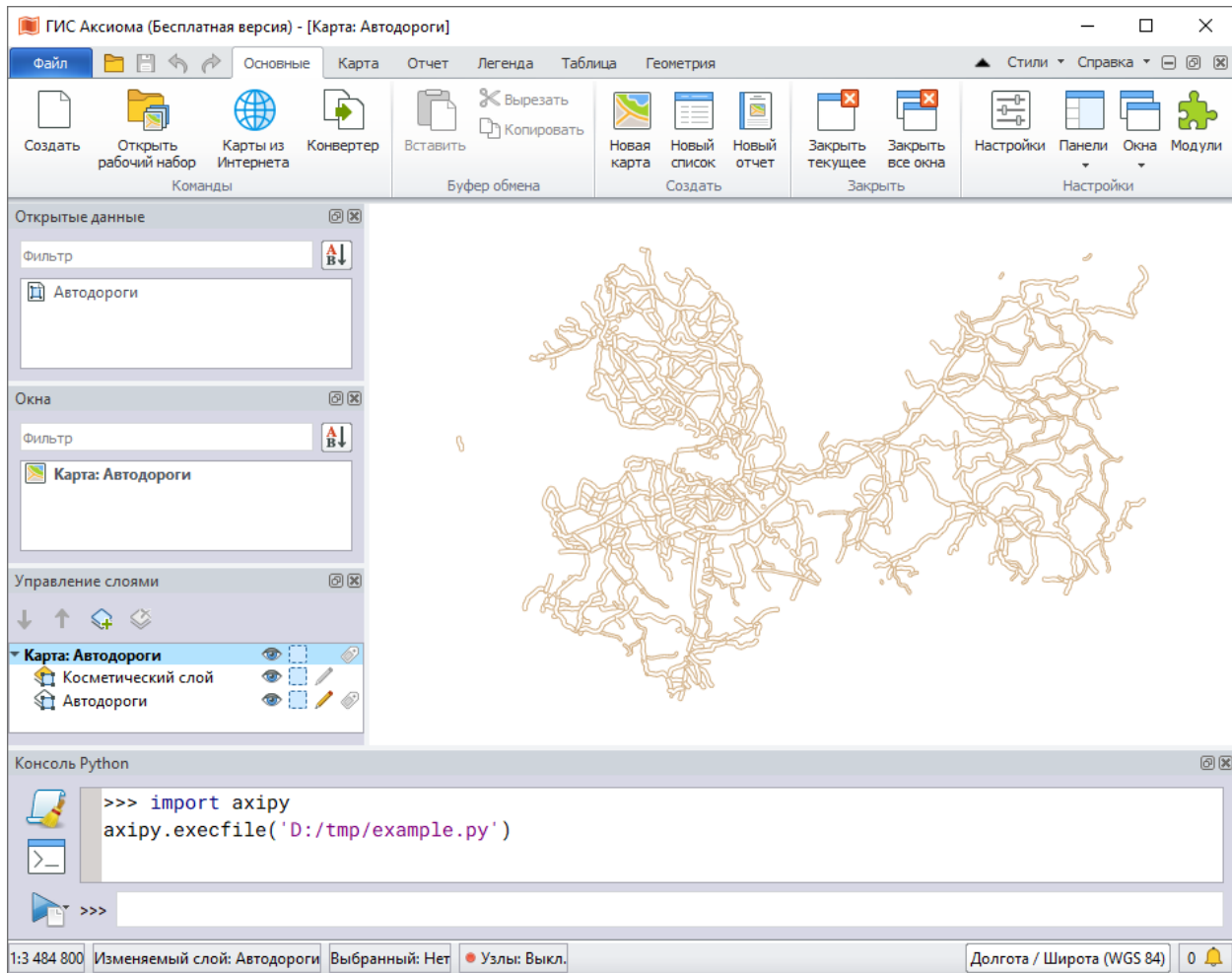
С помощью ахіру можно открыть географические данные и показать их на карте.

В примере используются данные с сайта Аксиомы «Обзорная карта Ленинградская область». Скачать данные можно по ссылке <https://axioma-gis.ru/download> (Раздел «Документация и данные»).

Для выполнения примера, нужно скопировать код ниже и выполнить его в редакторе кода Аксиомы.

```
from axipy import provider_manager, Layer, Map, view_manager, Rect, open_file_dialog

# Открытие диалога выбора файла
path_to_file = open_file_dialog("MapInfo Tab (*.tab)")
# Открытие файла
data_object = provider_manager.openfile(str(path_to_file))
# Создание слоя
layer = Layer.create(data_object)
# Создание карты
map_ = Map([layer])
# Создание окна карты
map_view = view_manager.create_mapview(map_)
```



С открытой картой можно взаимодействовать и из консоли Python, например, добавить точку в центр окна карты.

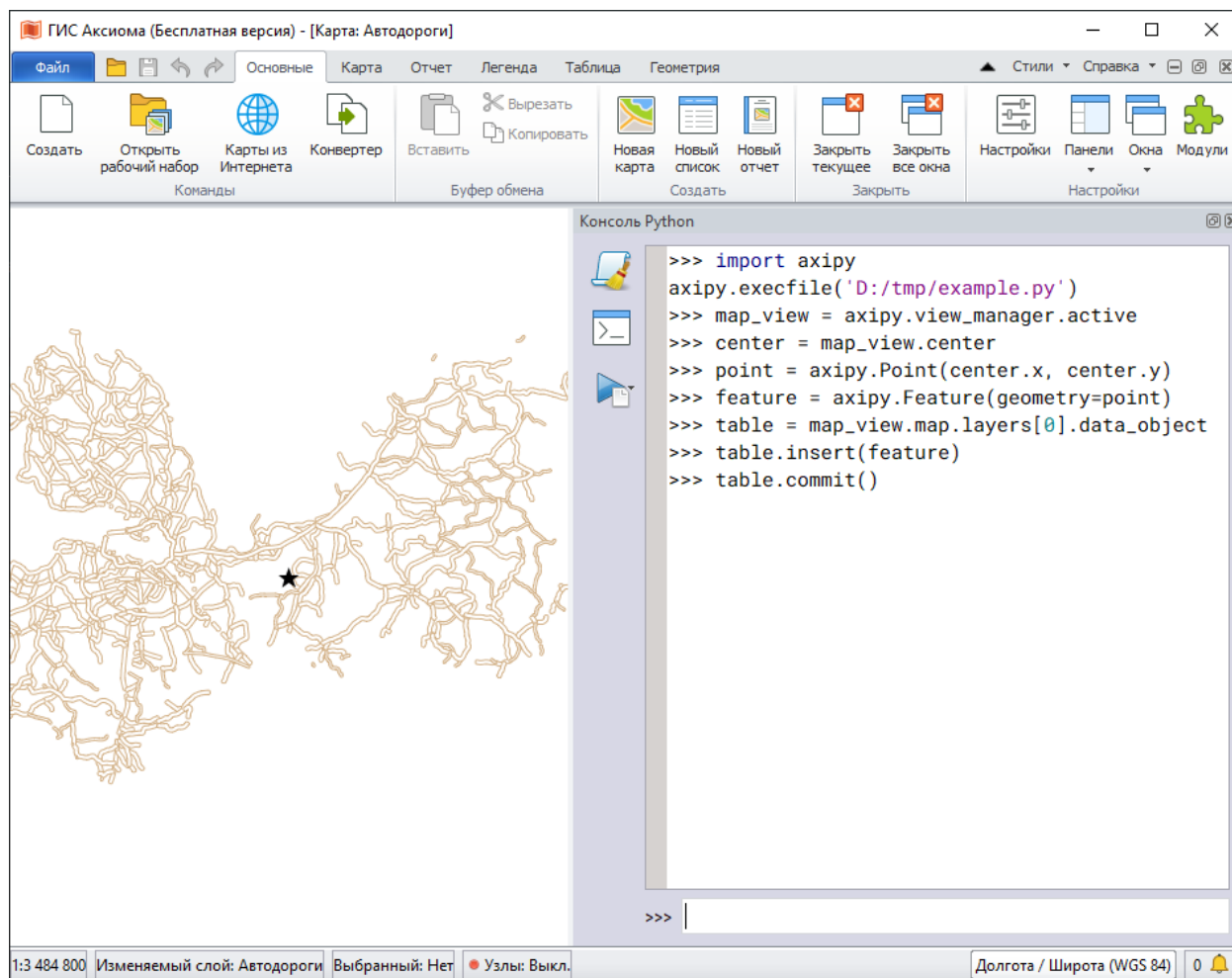
```
import axipy

# Получение активного окна карты
map_view = axipy.view_manager.active

# Получение координат центра окна карты
center = map_view.center
# Создание точки с полученными координатами
point = axipy.Point(center.x, center.y)
# Создание записи с точкой
feature = axipy.Feature(geometry=point)

# Получение таблицы, связанной с картой, через первый слой карты
table = map_view.map.layers[0].data_object
# Вставка записи с точкой в таблицу
table.insert(feature) # Теперь добавленная точка, отразится на карте\

# Чтобы сохранить изменения в таблице, нужно вызвать:
table.commit()
```



Использование среды разработки

При создании плагина или скрипта для ГИС Аксиома рекомендуется использовать интегрированные среды разработки (IDE). Среда разработки предоставляет полезные возможности, такие как: подсветка кода, автодополнение, отладка, система контроля версий и другие.

Для использования `axipy` из среды разработки необходимо вызвать функцию `axipy.init_axioma()`. Чтобы запустить Аксиому из среды разработки под режимом отладки нужно выполнить код:

```
from axipy import init_axioma, mainwindow

# Инициализация ядра
app = init_axioma()
# Создание и отображение главного окна Аксиомы
mainwindow.show()
# Запуск основного цикла приложения
app.exec_()
```

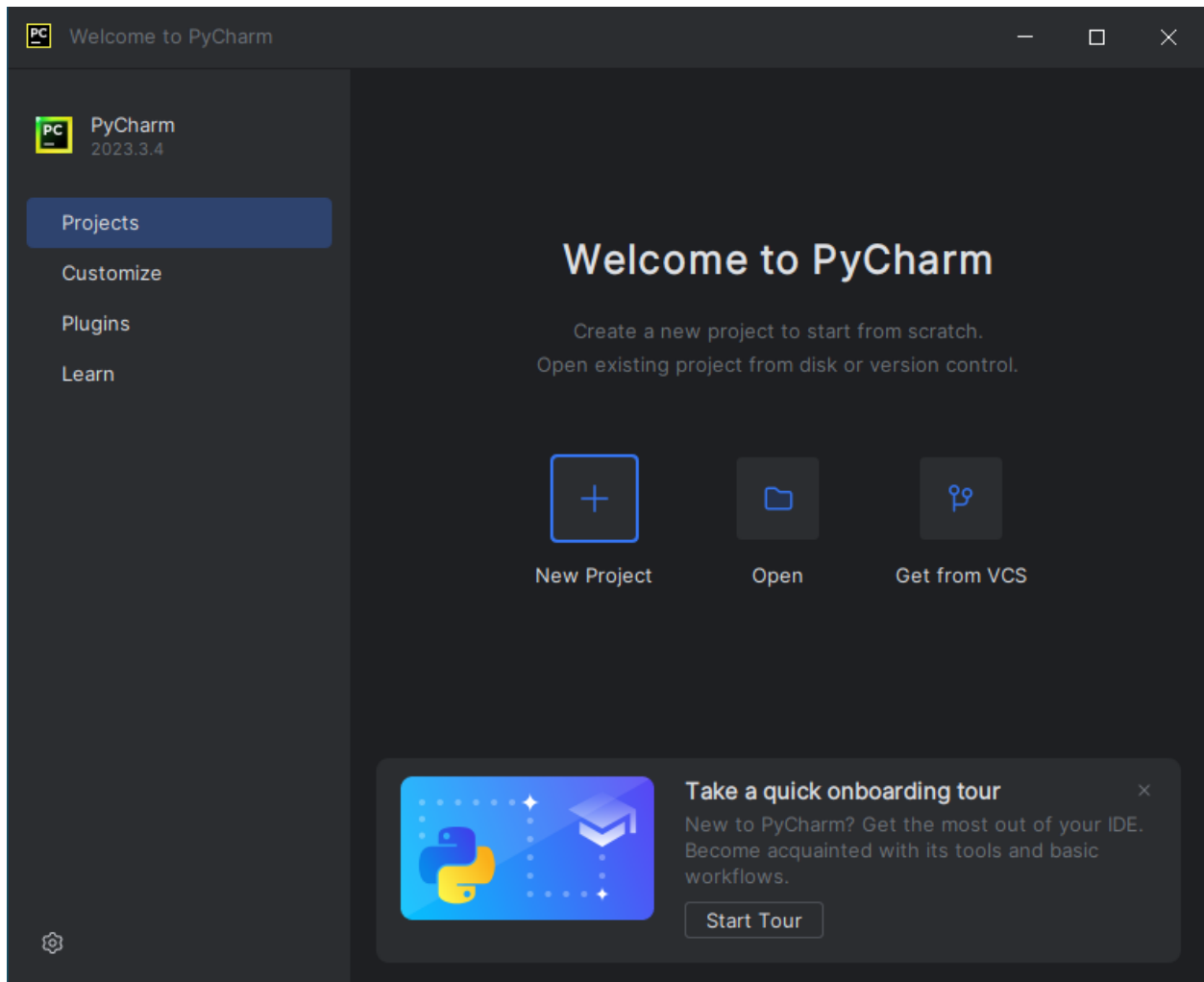
Одной из популярных сред разработки является PyCharm от JetBrains. У PyCharm есть две версии: платная PyCharm Professional и бесплатная PyCharm Community Edition. Скачать бесплатную версию PyCharm Community Edition можно по ссылке: <https://www.jetbrains.com/pycharm/download>.

3.1 PyCharm Windows

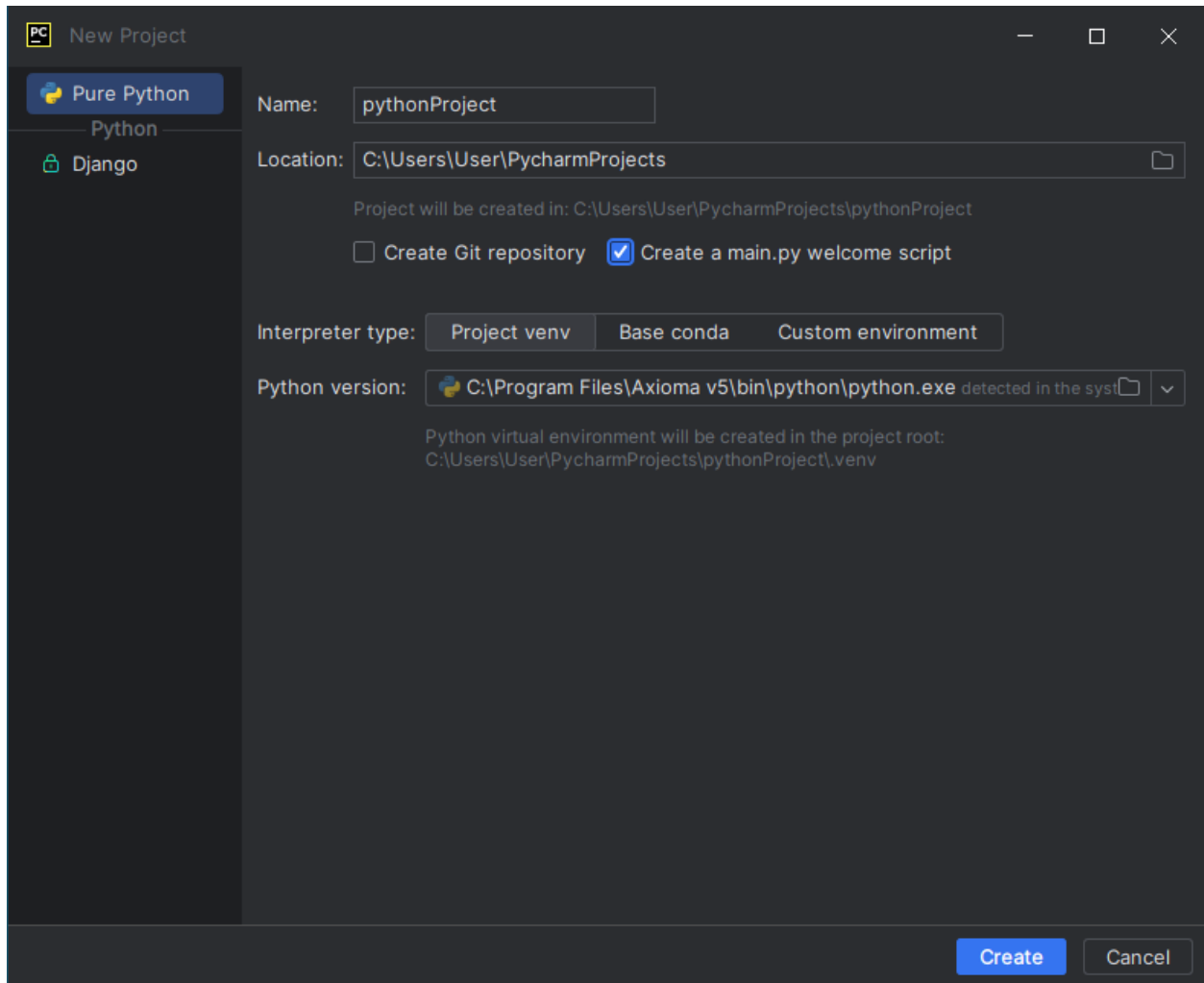
Інструкція написана для версії PyCharm 2023.3.4.

3.1.1 Создание проекта.

После запуска PyCharm нужно создать новый проект, нажав кнопку “New Project” в начальном окне.

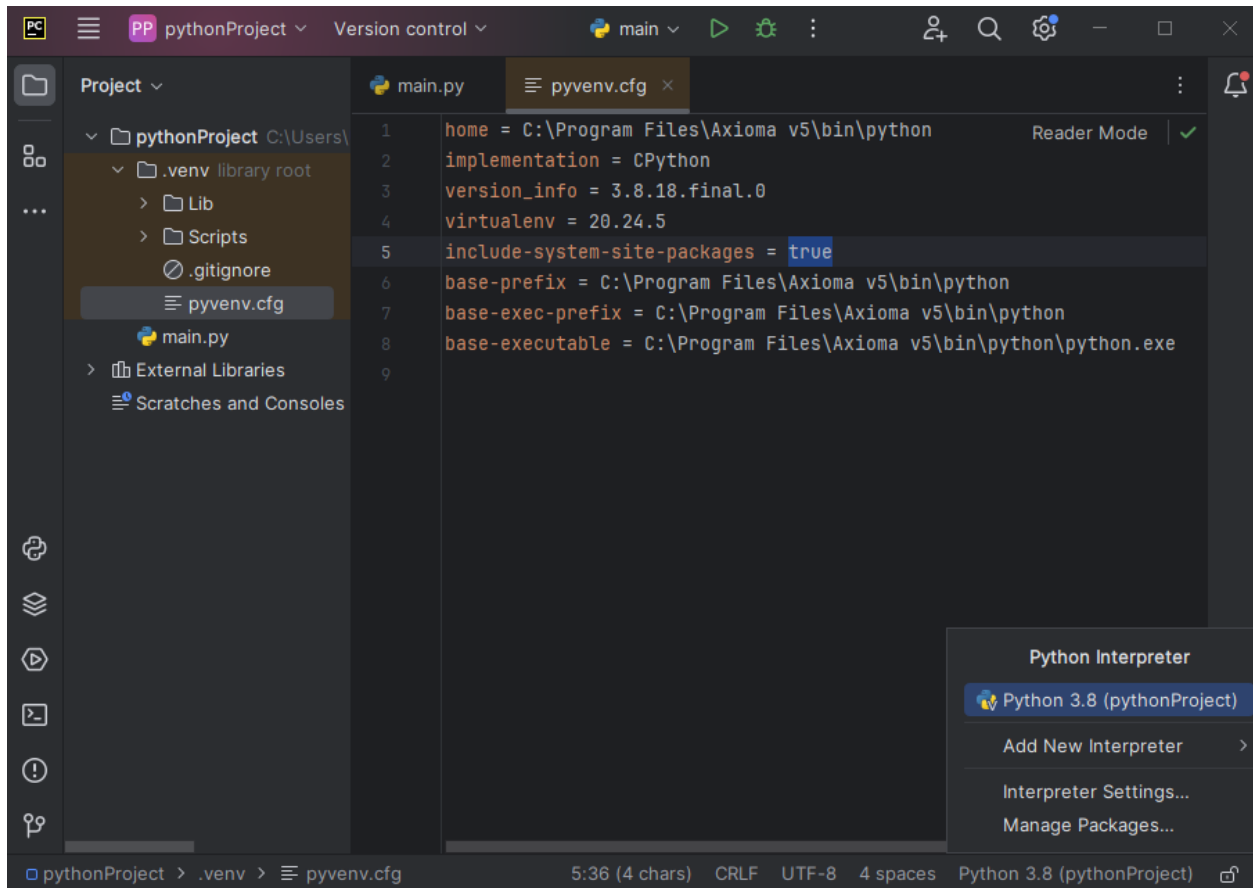


В настройках проекта, в поле «Location» следует указать путь к папке проекта. В этой папке будут храниться используемые файлы. Для работы с ахіру требуется создать новое виртуальное окружение Python. Для этого, в поле «Python version», нужно указать путь к Python ГИС Аксиома: C:\Program Files\Axioma v5\bin\python\python.exe. PyCharm автоматически создаст новую папку с виртуальным окружением (.venv) в папке проекта. Галочку «Create a main.py welcome script» можно оставить нажатой, для того чтобы PyCharm создал новый main.py файл.



3.1.2 Настройка виртуального окружения.

После создания проекта, нужно настроить виртуального окружение, чтобы оно могло обнаружить ахіру. Для этого, нужно дождаться построения проекта PyCharm, и открыть файл `.venv\pyvenv.cfg` на редактирование. В нем, нужно изменить значение параметра «include-system-site-packages» на «true». Чтобы изменение вступило в силу, нужно перевыбрать текущий интерпретатор проекта в правом нижнем углу и дождаться построения проекта PyCharm.



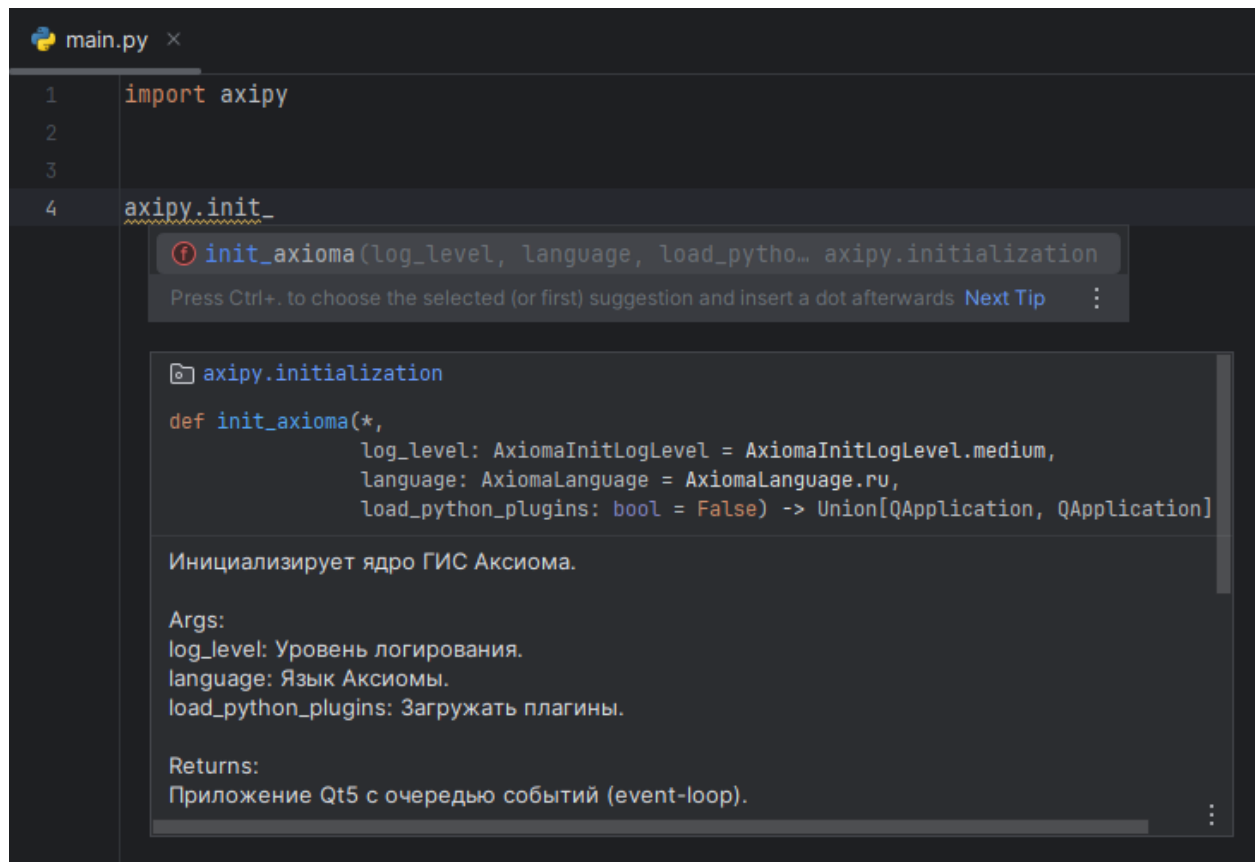
3.1.3 Настройка подсказок и запуск кода.

Чтобы проверить работу PyCharm с ахіру можно выполнить следующий код:

```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода PyCharm будет отображать подсказки. Чтобы сделать подсказки более подробными, нужно открыть окно «Settings» (Ctrl + Alt + S), перейти на вкладку Editor | General | Code Completion и отметить галочку «Show the documentation popup».



3.1.4 Разработка плагинов (модулей) для Аксиомы.

В папке C:\Program Files\Axioma v5\bin\python_plugins находятся встроенные модули Аксиомы. Их можно использовать как пример при написании плагинов.

Чтобы начать разработку плагина, можно скопировать минимальный пример модуля `ru_axioma_gis_axipy_example_plugin_minimal` в папку проекта.

```
pythonProject # папка с проектом
├── ru_axioma_gis_axipy_example_plugin_minimal # модуль
│   ├── __init__.py # Точка входа модуля
│   └── manifest.ini # Описание модуля
```

Далее, нужно переименовать папку с модулем, например `example_plugin_minimal` (Название папки является уникальным идентификатором плагина). В файле `manifest.ini` нужно изменить отображаемое имя, чтобы найти плагин в списке модулей Аксиомы, например: `name=Минимальный пользовательский плагин`

Затем, нужно запустить Аксиому и добавить папку проекта в Дополнительные пути загрузки модулей. (Вкладка Основные | Настройки | Модули | Дополнительные модули | Настройки) Далее, нужно включить плагин в списке установленных плагинов. Теперь, при каждом запуске Аксиомы, плагин будет автоматически загружаться.

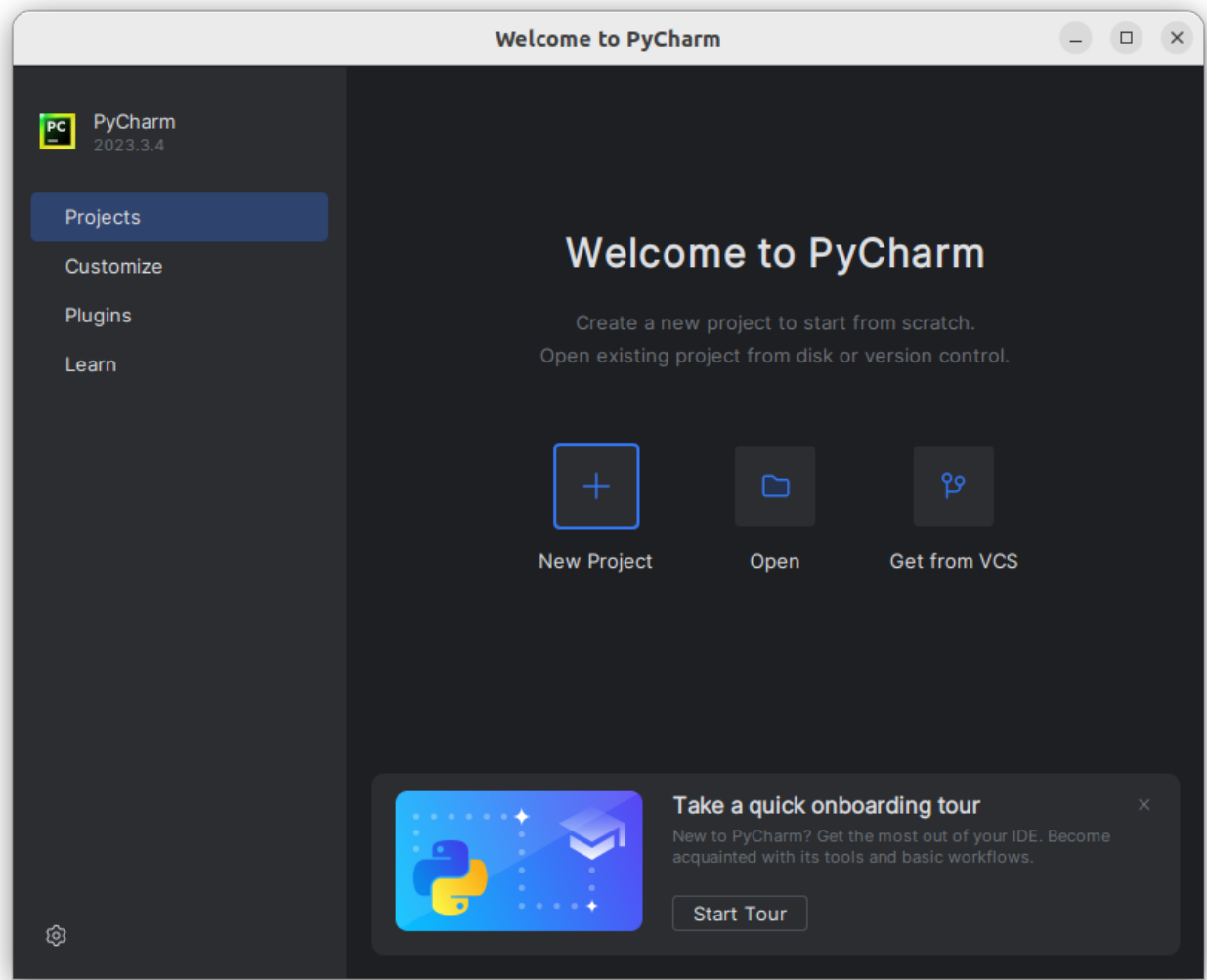
Подробнее о разработке плагинов можно прочитать в разделе [Разработка Плагинов \(Модулей\)](#).

3.2 PyCharm Linux

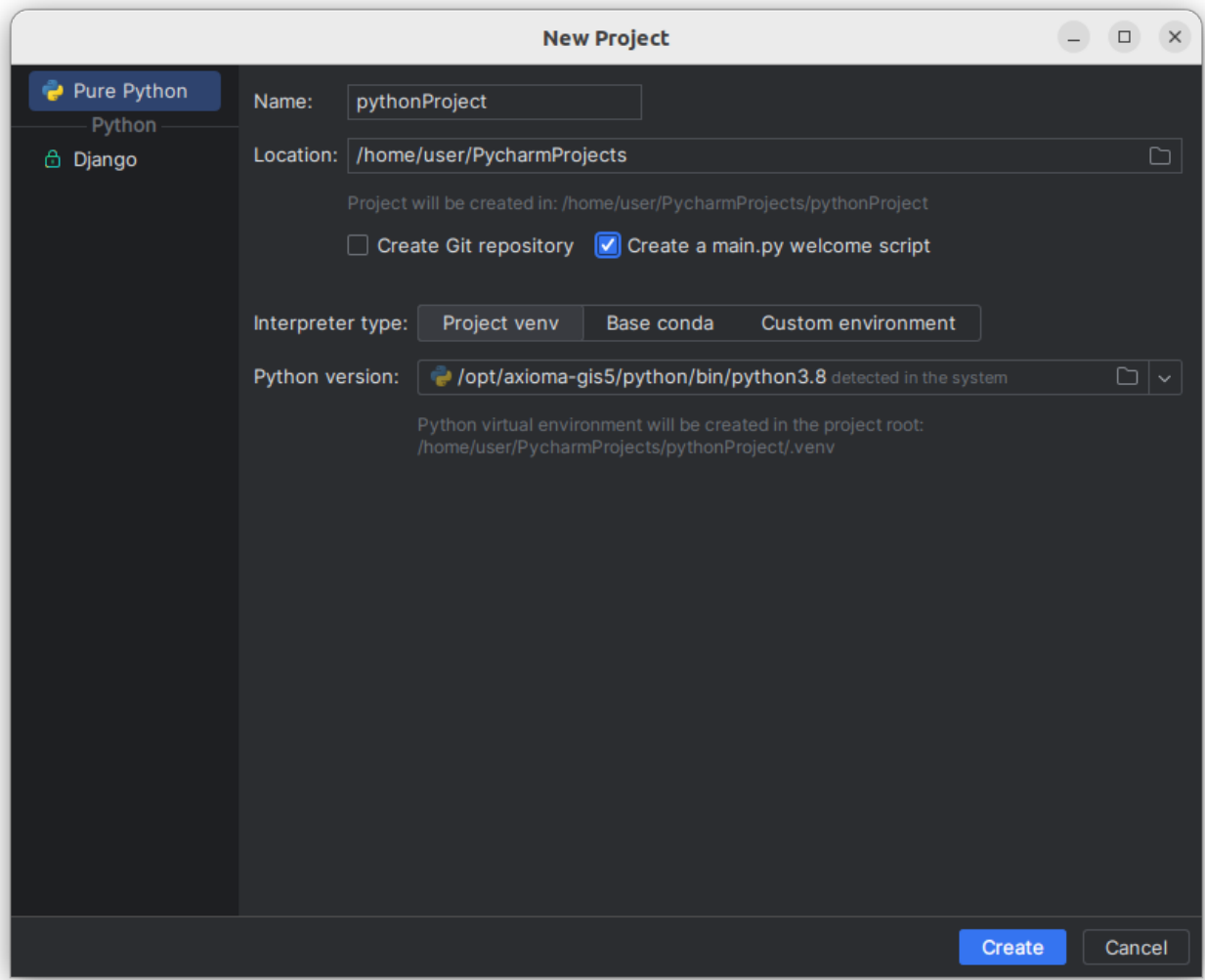
Інструкція написана для версії PyCharm 2023.3.4.

3.2.1 Створення проекту.

Після запуску PyCharm потрібно створити новий проект, нажав кнопку “New Project” в початковому вікні.

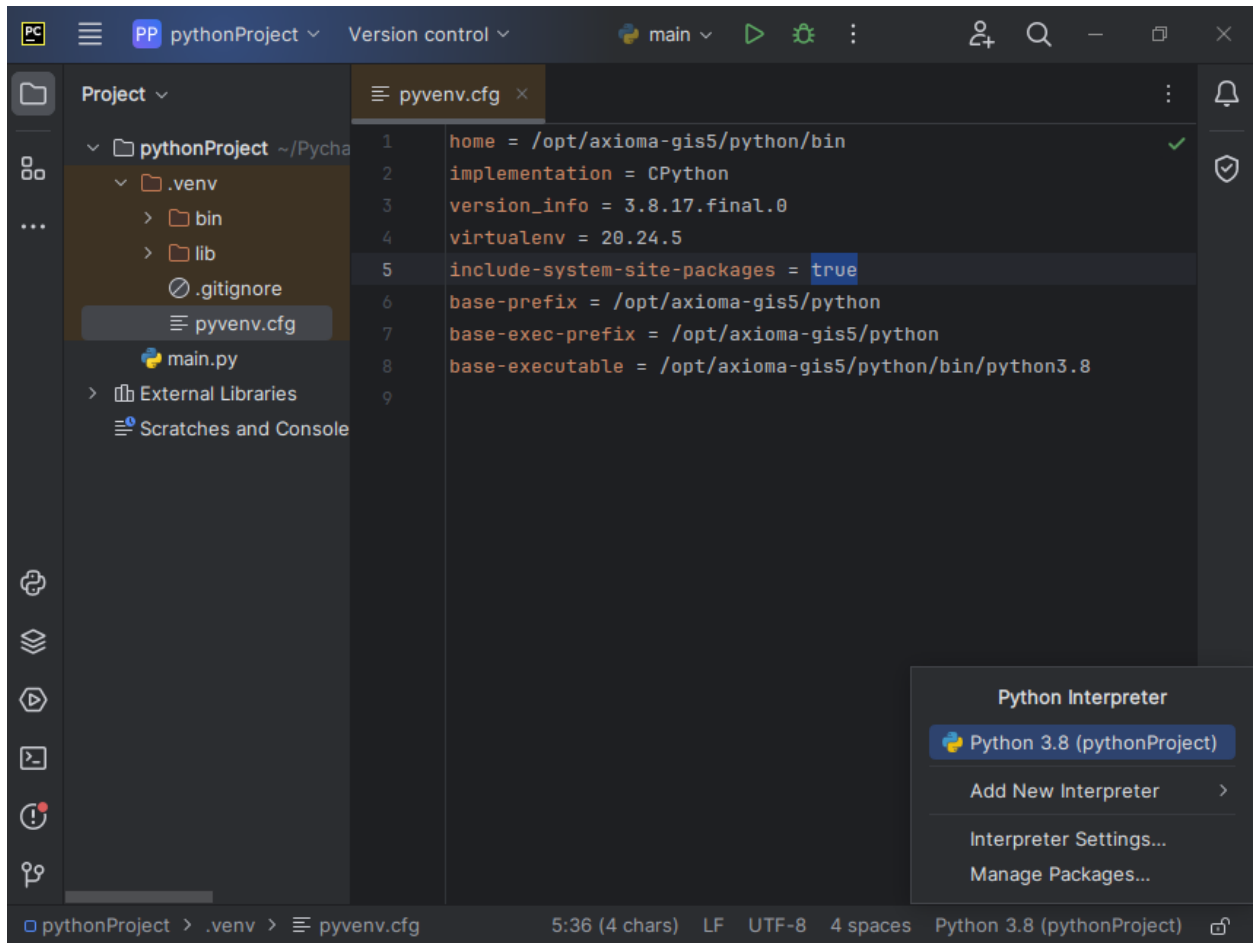


В налаштуваннях проекту, в полі «Location» слід вказати шлях до папки проекту. В цій папці будуть зберігатися використовувані файли. Для роботи з ахіру потрібно створити нове віртуальне середовище Python. Для цього, в полі «Python version», потрібно вказати шлях до Python ГІС Аксиома: /opt/axioma-gis5/python/bin/python3.8. PyCharm автоматично створить нову папку з віртуальним середовищем (.venv) в папці проекту. Галочку «Create a main.py welcome script» можна залишити натиснутою, для того щоб PyCharm створив новий main.py файл.



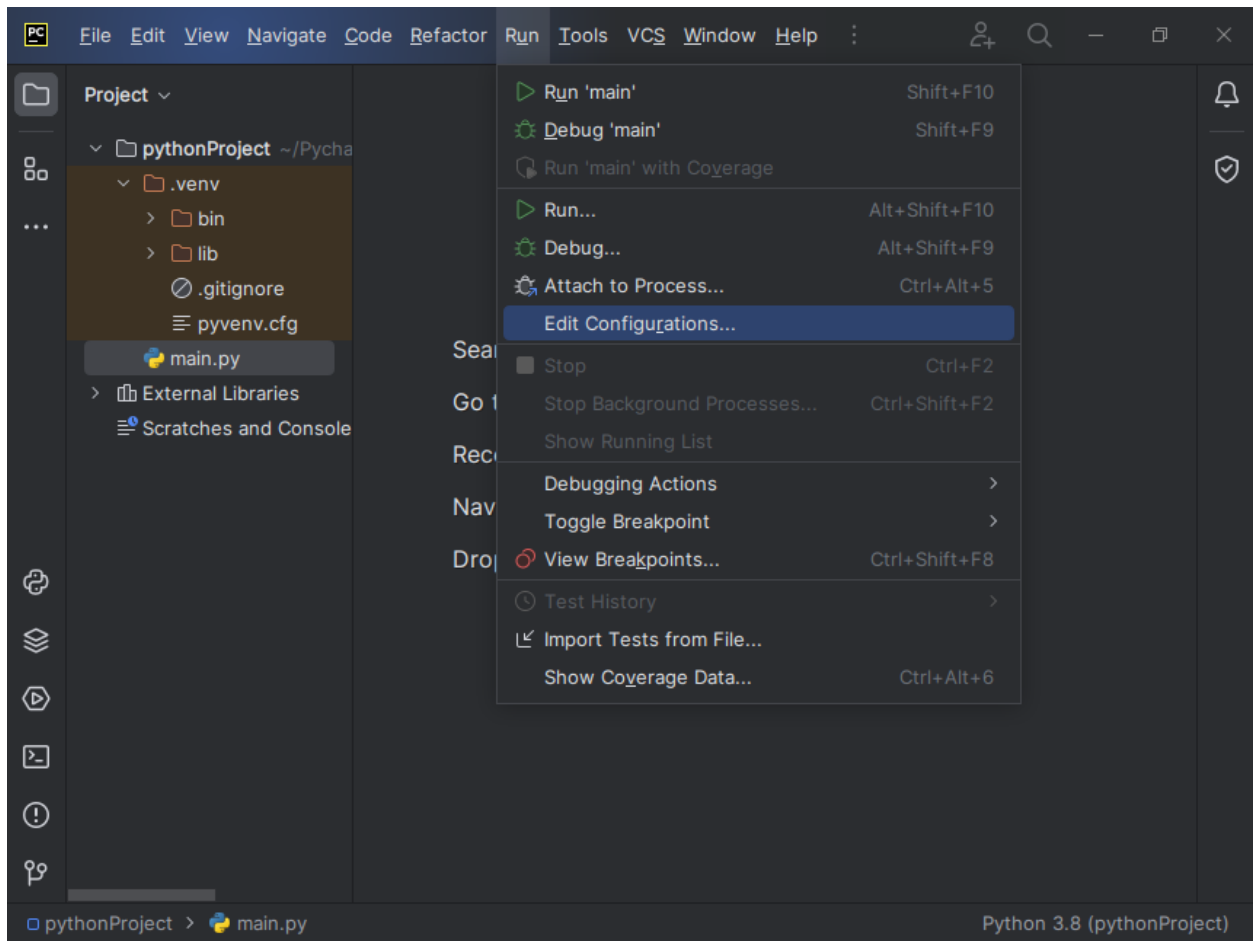
3.2.2 Настройка виртуального окружения.

После создания проекта, нужно настроить виртуального окружение, чтобы оно могло обнаружить ахіру. Для этого, нужно дождаться построения проекта PyCharm, и открыть файл `.venv/pyvenv.cfg` на редактирование. В нем, нужно изменить значение параметра «include-system-site-packages» на «true». Чтобы изменение вступило в силу, нужно перевыбрать текущий интерпретатор проекта в правом нижнем углу и дождаться построения проекта PyCharm.

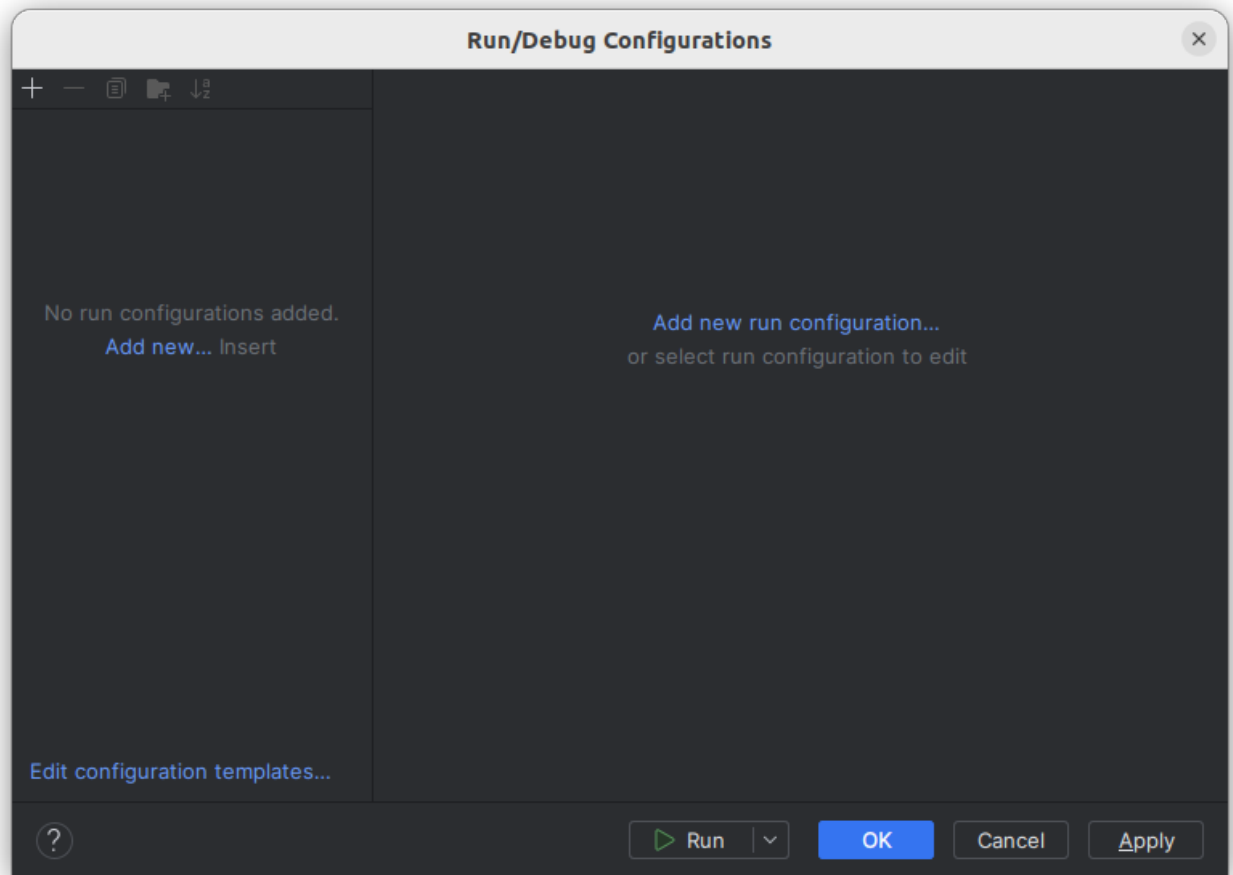


3.2.3 Настройка переменных среды.

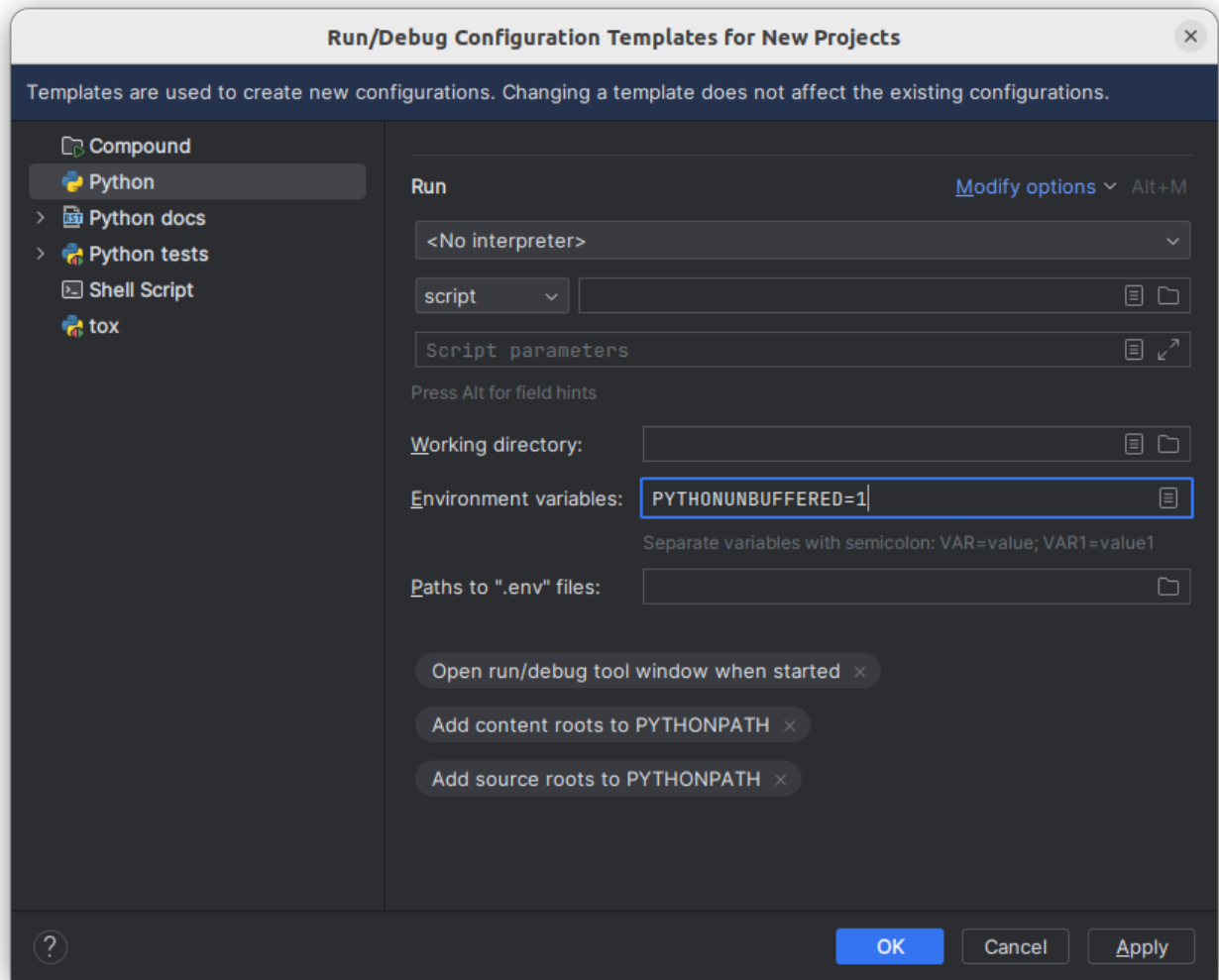
Дальше, нужно настроить переменную среды, для того чтобы PyCharm смог обнаружить динамические библиотеки ахіру. После окончательного построения проекта PyCharm, нужно вызвать из меню «Run – Edit Configuration...».



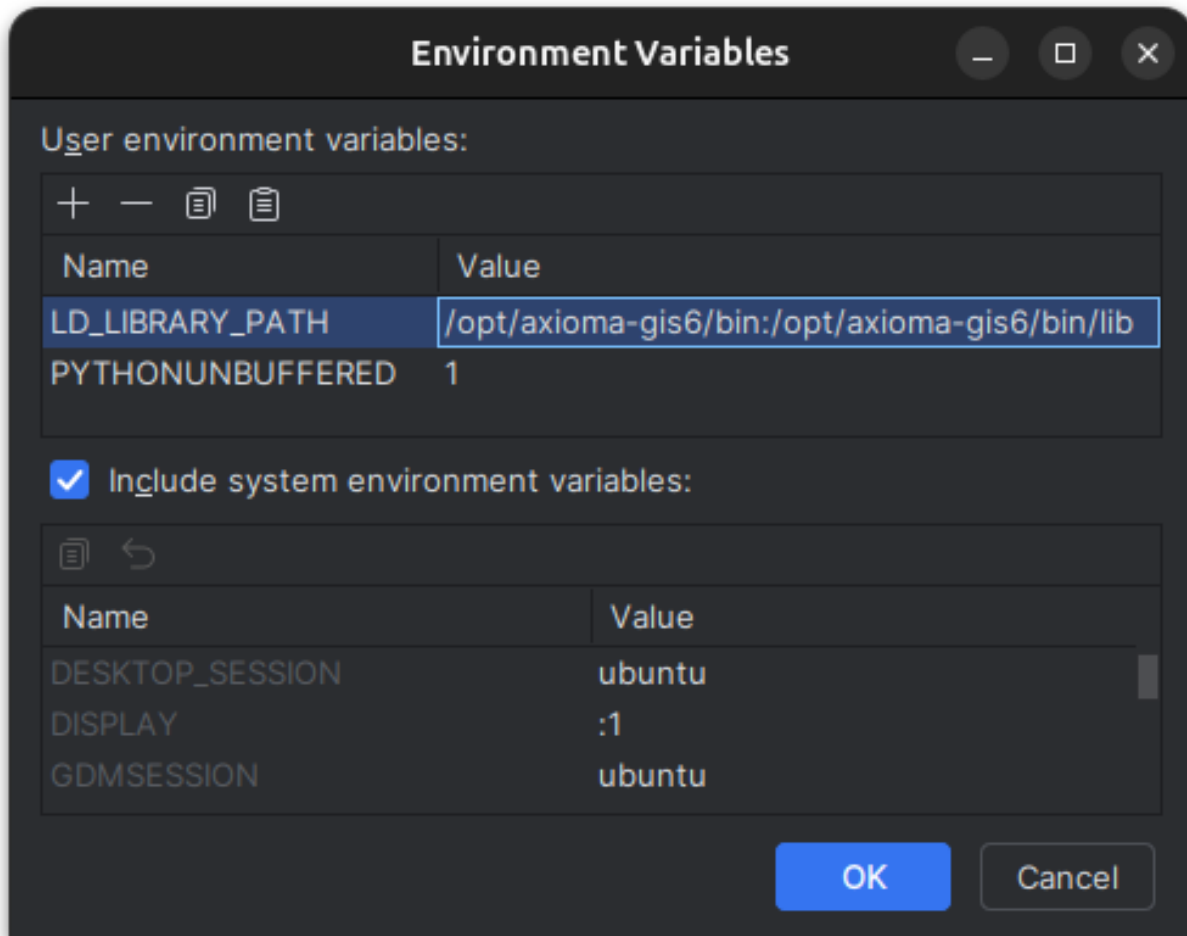
После появления окна настроек конфигураций следует удалить существующие конфигурации в левой части окна (если таковые имеются). А затем нажать «Edit configuration templates...» в нижней левой части окна.



В появившемся окне нужно добавить переменную окружения `LD_LIBRARY_PATH` с указанием на папки «`/opt/axioma-gis6/bin:/opt/axioma-gis6/bin/lib`».



Для этого нужно внести изменение в поле «Environment variables», нажав на кнопку справа от поля.



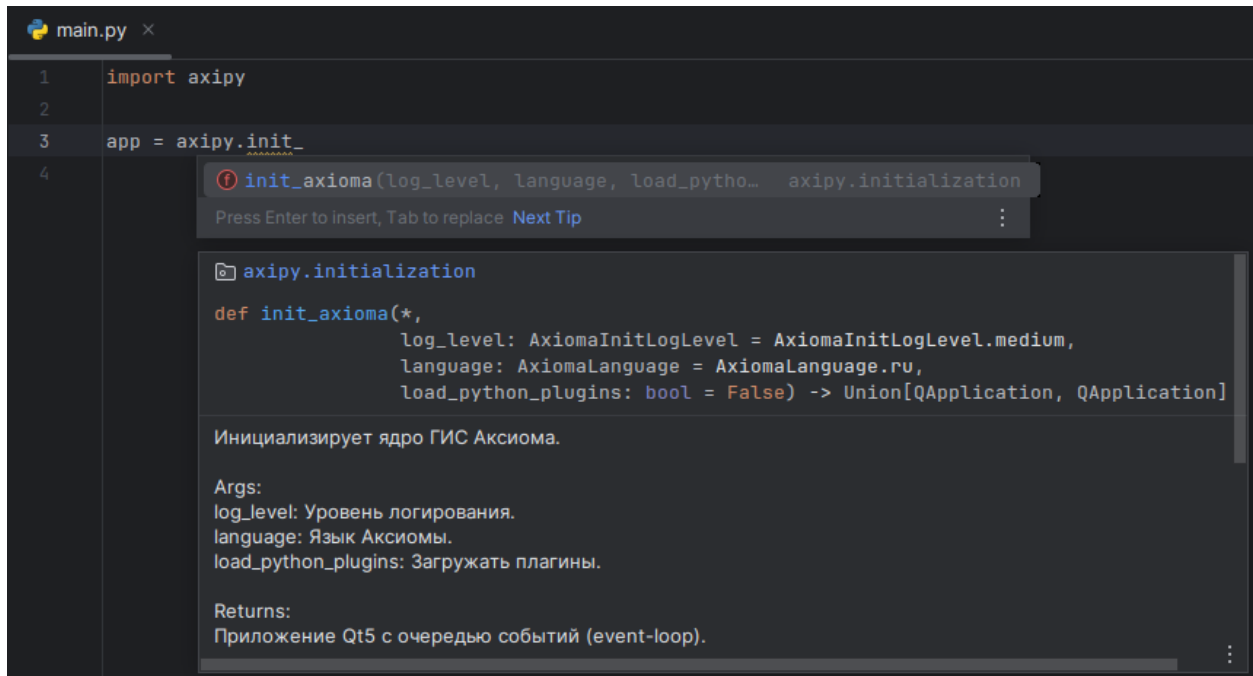
3.2.4 Настройка подсказок и запуск кода.

Чтобы проверить работу PyCharm с ахіру можно выполнить следующий код:

```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода PyCharm будет отображать подсказки. Чтобы сделать подсказки более подробными, нужно открыть окно «Settings» (Ctrl + Alt + S), перейти на вкладку Editor | General | Code Completion и отметить галочку «Show the documentation popup».



3.2.5 Разработка плагинов (модулей) для Аксиомы.

В папке `/opt/axioma-gis5/bin/python_plugins/` находятся встроенные модули Аксиомы. Их можно использовать как пример при написании плагинов.

Чтобы начать разработку плагина, можно скопировать минимальный пример модуля `ru_axioma_gis_axipy_example_plugin_minimal` в папку проекта.

```

pythonProject # папка с проектом
├── ru_axioma_gis_axipy_example_plugin_minimal # модуль
│   ├── __init__.py # Точка входа модуля
│   └── manifest.ini # Описание модуля

```

Далее, нужно переименовать папку с модулем, например `example_plugin_minimal` (Название папки является уникальным идентификатором плагина). В файле `manifest.ini` нужно изменить отображаемое имя, чтобы найти плагин в списке модулей Аксиомы, например: `name=Минимальный пользовательский плагин`

Затем, нужно запустить Аксиому и добавить папку проекта в Дополнительные пути загрузки модулей. (Вкладка Основные | Настройки | Модули | Дополнительные модули | Настройки) Далее, нужно включить плагин в списке установленных плагинов. Теперь, при каждом запуске Аксиомы, плагин будет автоматически загружаться.

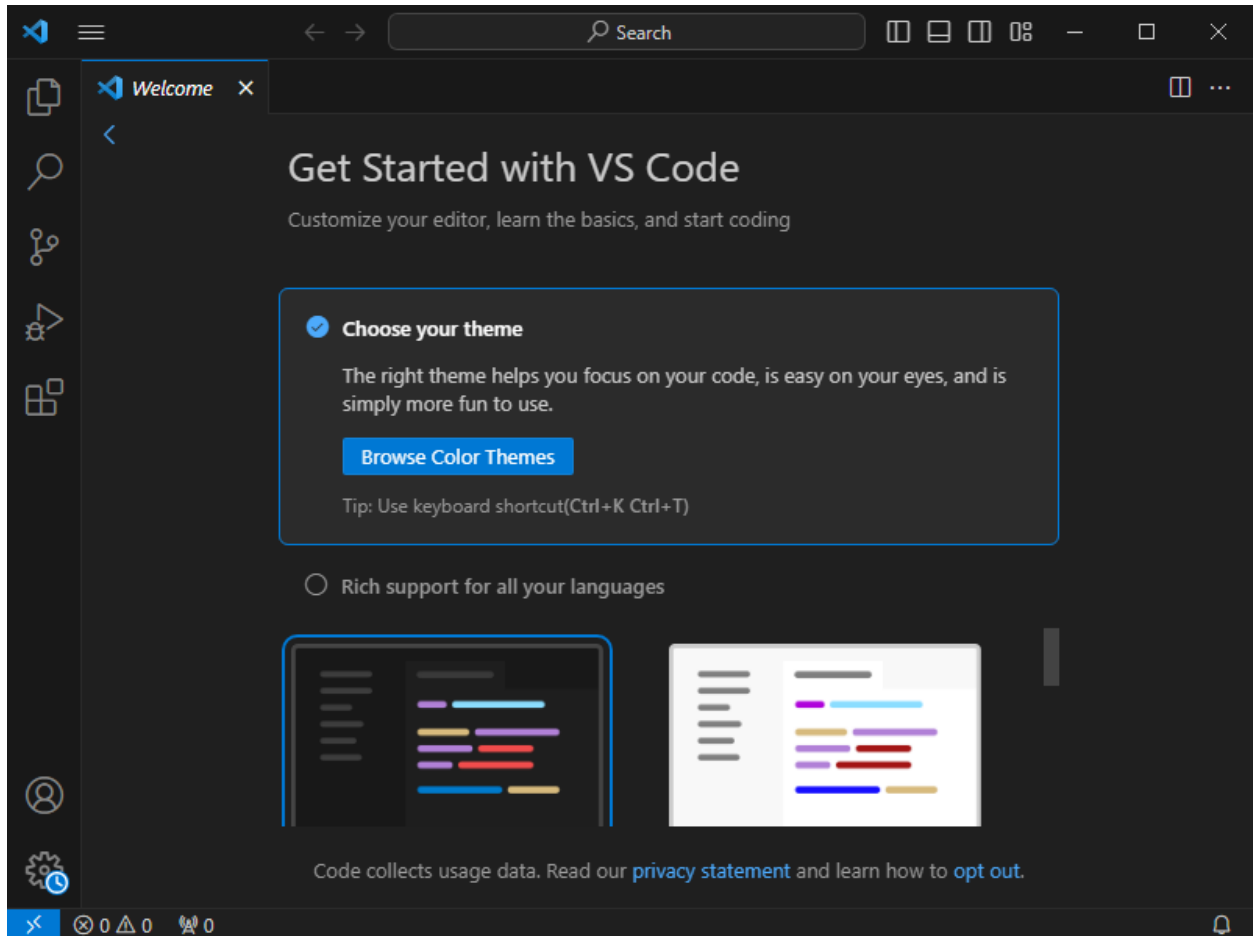
Подробнее о разработке плагинов можно прочитать в разделе [Разработка Плагинов \(Модулей\)](#).

Другая популярная среда разработки Visual Studio Code (VS Code) от MicroSoft. VS Code бесплатна и скачать ее можно по ссылке: <https://code.visualstudio.com>.

3.3 VS Code Windows

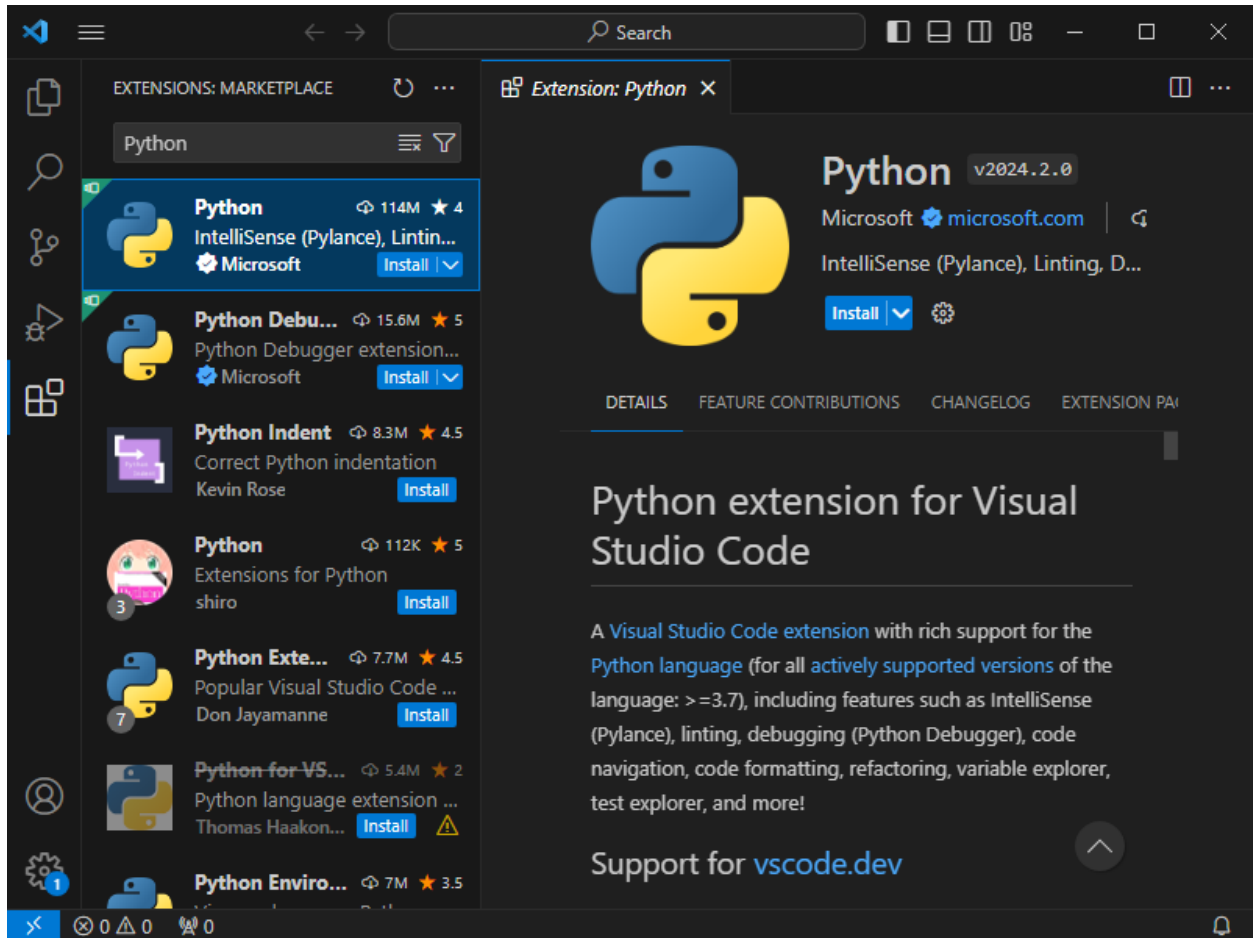
Інструкція написана для версії VS Code 1.86.2.

3.3.1 Установка расширений.



После запуска VS Code, нужно установить расширение для работы с python. Для этого нужно выбрать из левого бокового меню, вкладку «Extensions» (Ctrl+Shift+X), набрать в поиске «Python» и установить расширение.

3.3.2 Создание проекта.



Далее, нужно создать папку проекта, например: `C:\Users\User\VSCodeProjects\pythonProject`.

В меню «File», пункт «Open folder...» (Ctrl+O). Нужно выбрать ранее созданную папку.

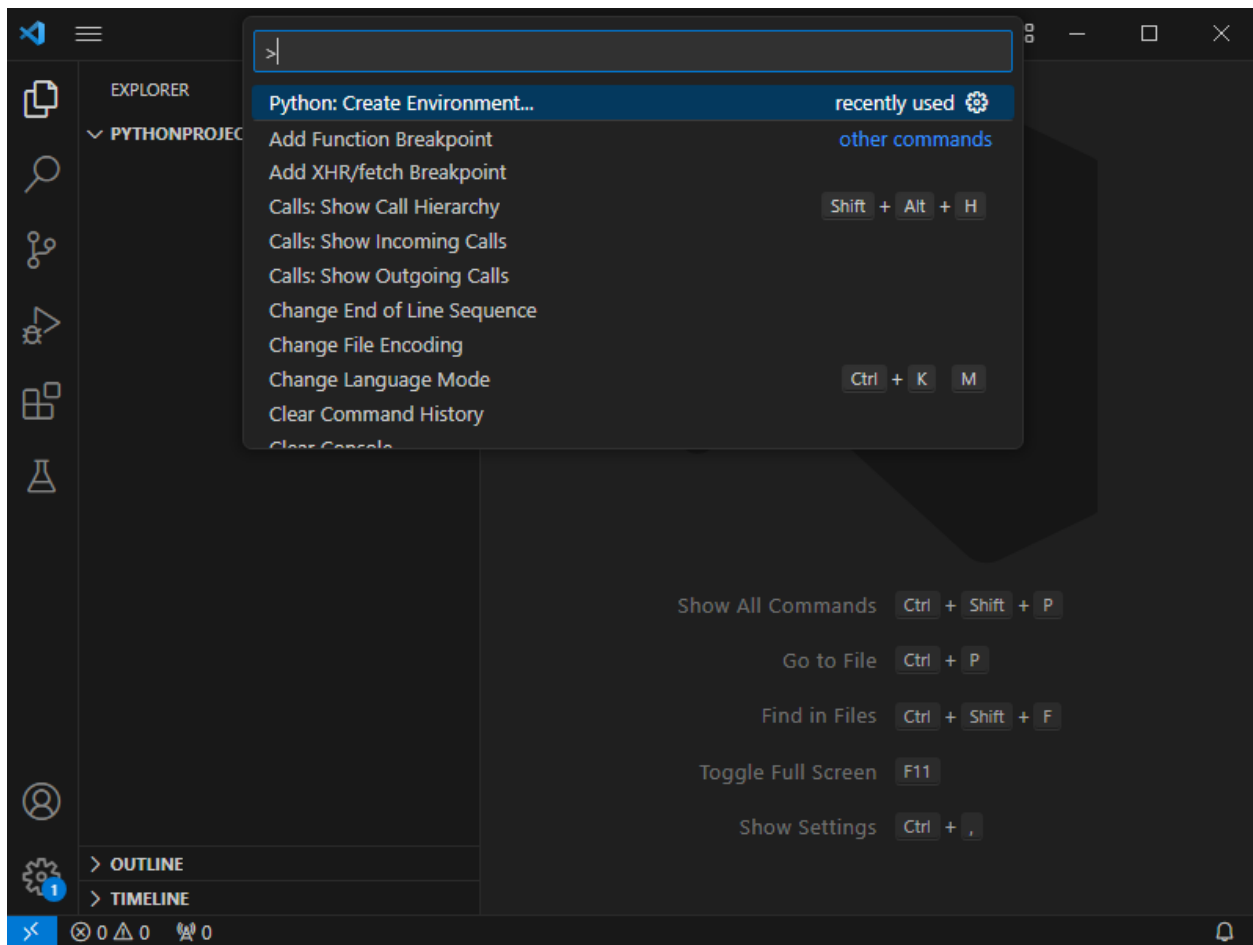
3.3.3 Создание виртуального окружения.

Сначала нужно открыть менеджер команд (Ctrl+Shift+P) «Command Palette».

Затем нужно выбрать команду Python: Create Environment... и выбрать вариант Venv. Далее, нужно ввести путь к питону Аксиомы

в поле Enter interpreter path...: `C:\Program Files\Axioma`

`v5\bin\python\python.exe`. Дождаться когда VS Code создаст виртуальное окружение.



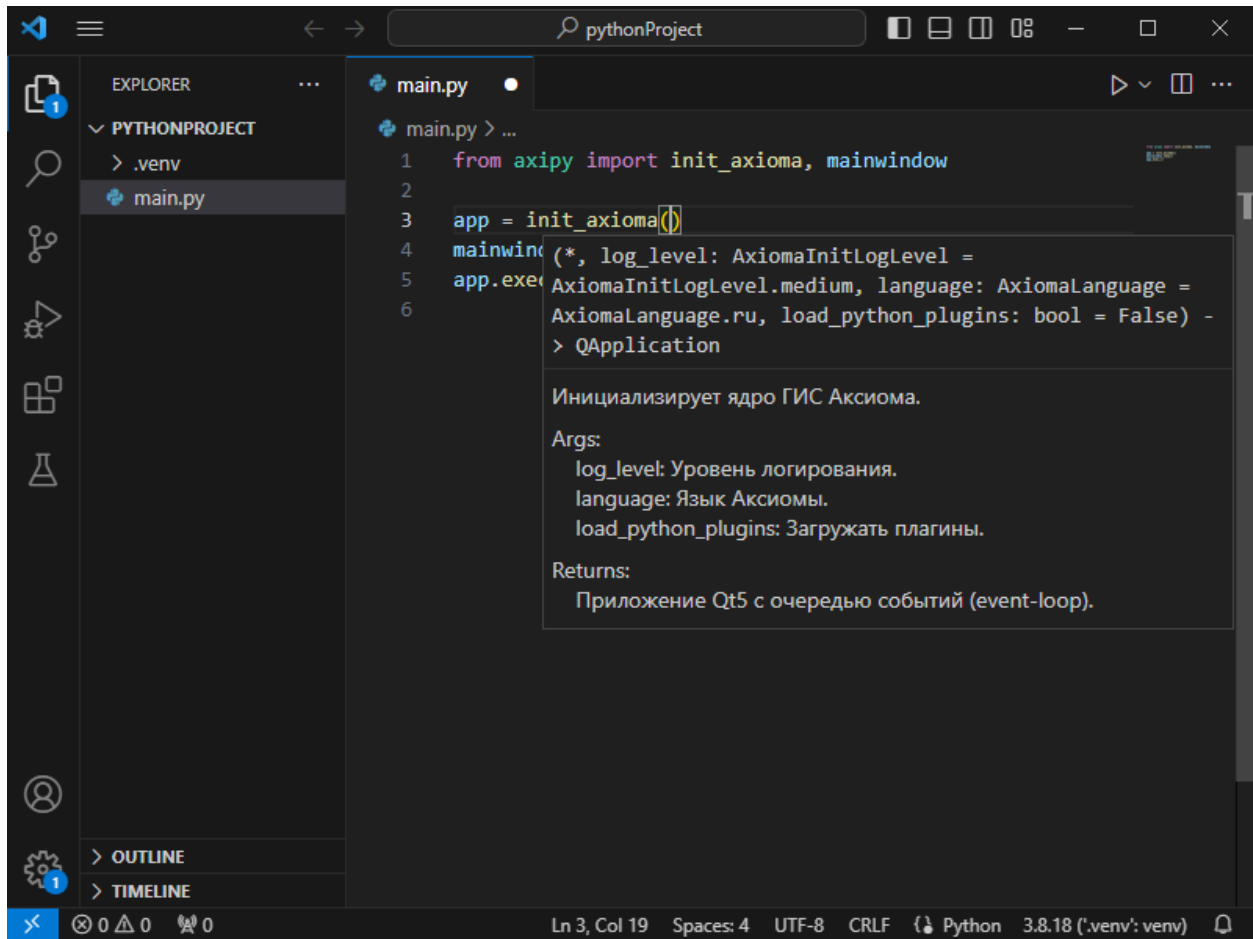
3.3.4 Запуск кода.

Чтобы проверить работу VS Code с axipy, нужно создать файл `main.py` в папке проекта и выполнить следующий код:

```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода VS Code будет отображать подсказки.



3.3.5 Разработка плагинов (модулей) для Аксиомы.

В папке C:\Program Files\Axiooma v5\bin\python_plugins находятся встроенные модули Аксиомы. Их можно использовать как пример при написании плагинов.

Чтобы начать разработку плагина, можно скопировать минимальный пример модуля ru_axioma_gis_axipy_example_plugin_minimal в папку проекта.

```

pythonProject # папка с проектом
├── ru_axioma_gis_axipy_example_plugin_minimal # модуль
│   ├── __init__.py # Точка входа модуля
│   └── manifest.ini # Описание модуля

```

Далее, нужно переименовать папку с модулем, например example_plugin_minimal (Название папки является уникальным идентификатором плагина). В файле manifest.ini нужно изменить отображаемое имя, чтобы найти плагин в списке модулей Аксиомы, например: name=Минимальный пользовательский плагин

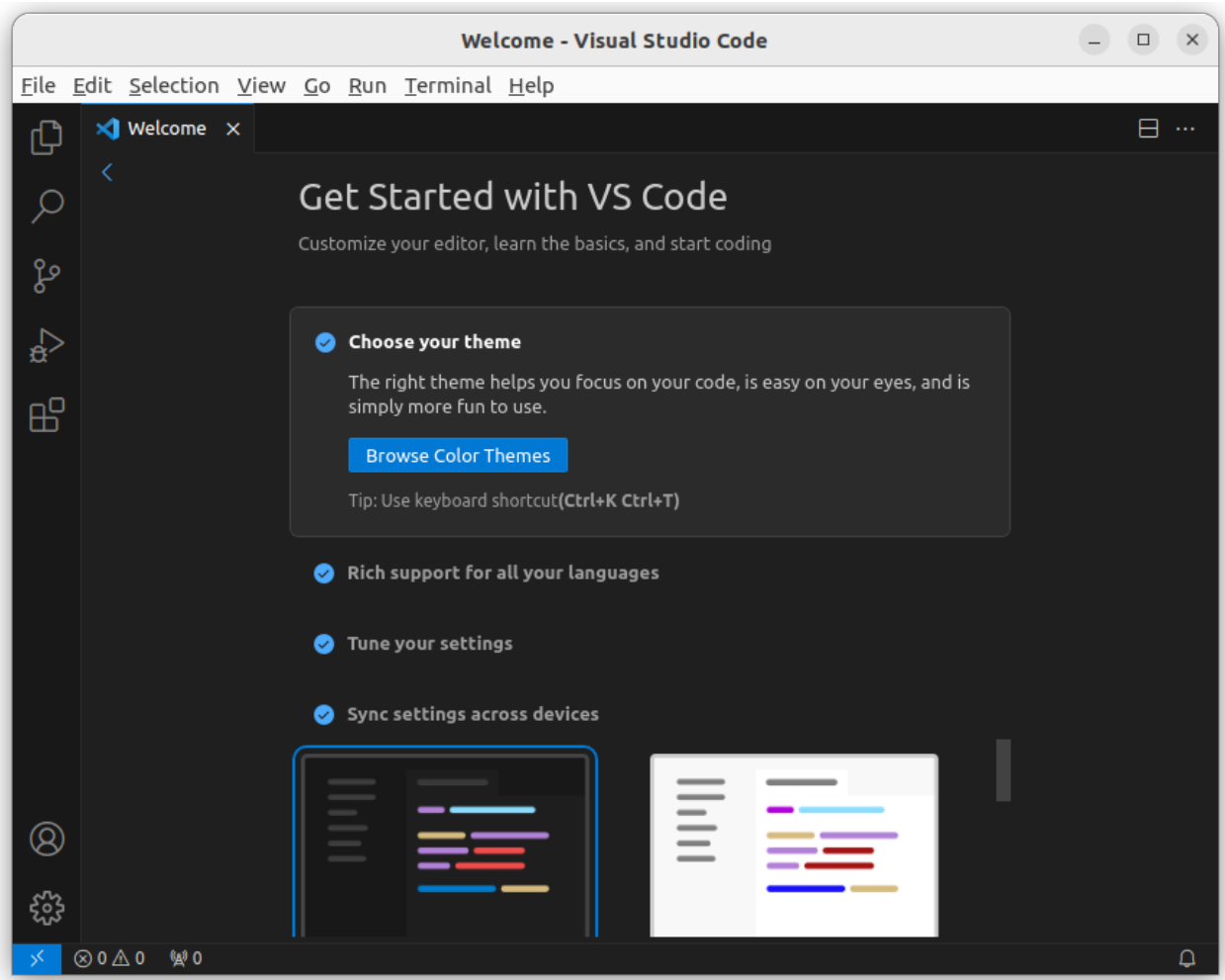
Затем, нужно запустить Аксиому и добавить папку проекта в Дополнительные пути загрузки модулей. (Вкладка Основные | Настройки | Модули | Дополнительные модули | Настройки) Далее, нужно включить плагин в списке установленных плагинов. Теперь, при каждом запуске Аксиомы, плагин будет автоматически загружаться.

Подробнее о разработке плагинов можно прочитать в разделе [Разработка Плагинов \(Модулей\)](#).

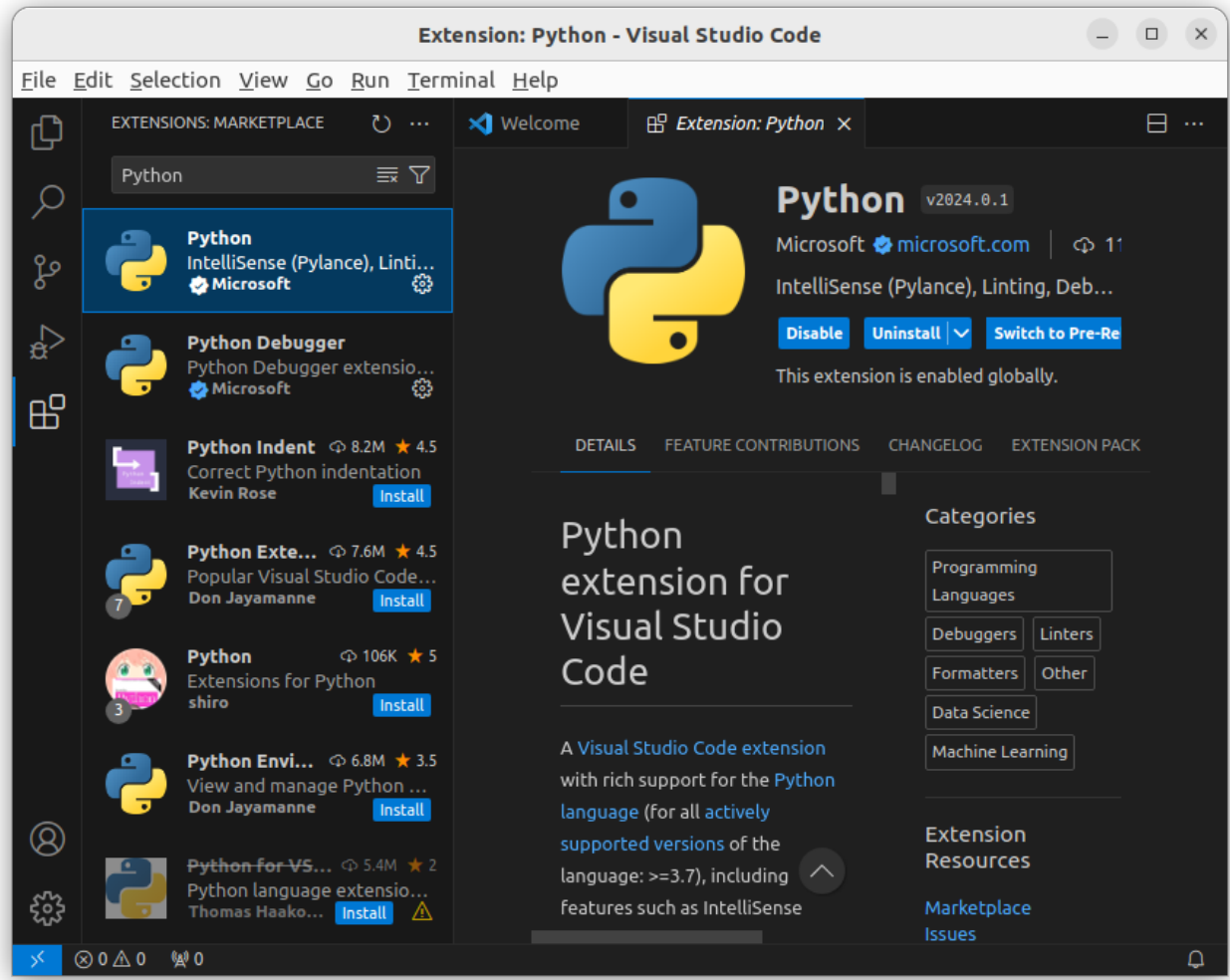
3.4 VS Code Linux

Инструкция написана для версии VS Code 1.86.2.

3.4.1 Установка расширений.



После запуска VS Code, нужно установить расширение для работы с python. Для этого нужно выбрать из левого бокового меню, вкладку «Extensions» (Ctrl+Shift+X), набрать в поиске «Python» и установить расширение.



3.4.2 Создание проекта.

Сначала, нужно создать папку проекта, например: `~/VSCodeProjects/pythonProject`.

Затем, в меню «File», нужно выбрать пункт «Open folder...» (Ctrl+O) и открыть ранее созданную папку.

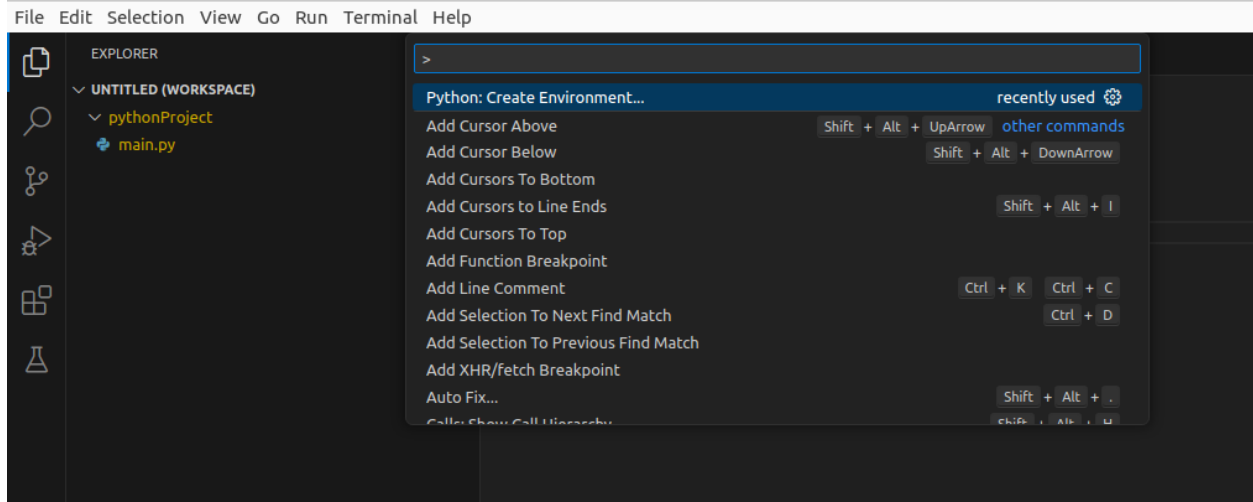
3.4.3 Создание виртуального окружения.

Сначала нужно открыть менеджер команд (Ctrl+Shift+P) «Command Palette».

Затем нужно выбрать команду Python: Create Environment... и выбрать вариант Venv. Далее, нужно ввести путь к питону Аксиомы

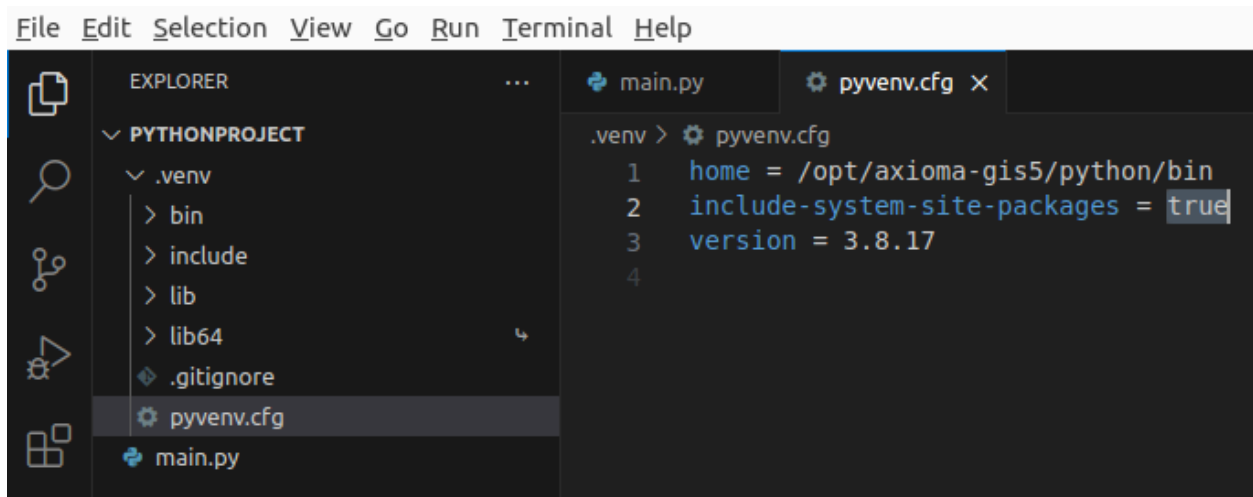
в поле Enter interpreter path...: `/opt/axioma-gis5/python/bin/python3.8`.

Дождаться когда VS Code создаст виртуальное окружение.



3.4.4 Настройка виртуального окружения.

После создания виртуального окружения, его нужно настроить, чтобы оно могло обнаружить ахіру. Для этого, нужно открыть файл `.venv/pyvenv.cfg` на редактирование. В нем, нужно изменить значение параметра «`include-system-site-packages`» на «`true`». Чтобы изменение вступило в силу, необходимо перезапустить VS Code.



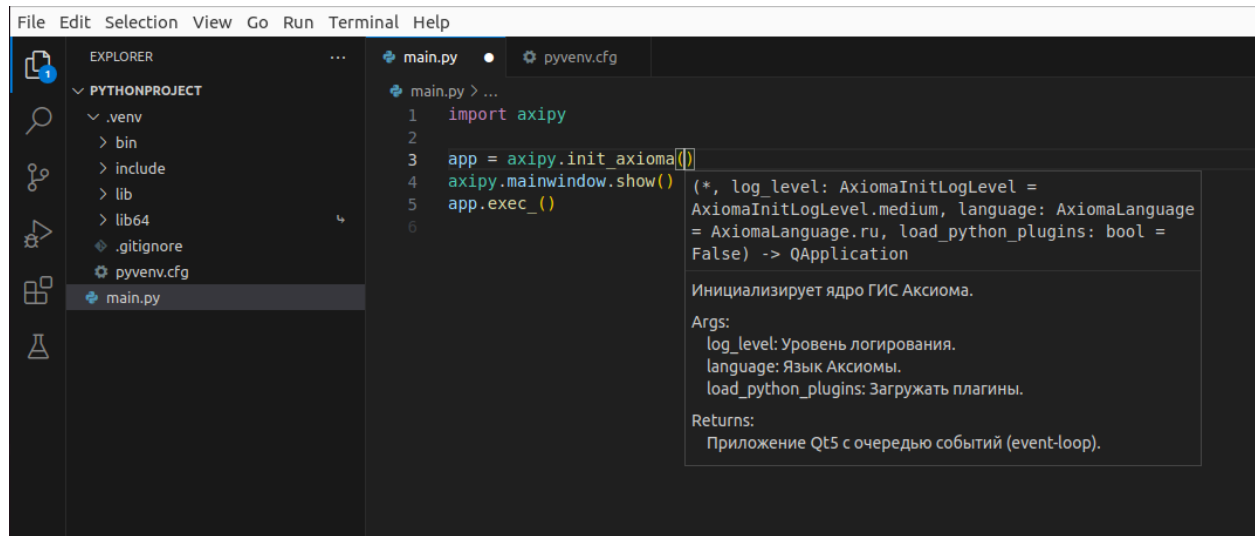
3.4.5 Запуск кода.

Чтобы проверить работу VS Code с ахіру, нужно создать файл `main.py` в папке проекта и выполнить следующий код:

```
from axipy import init_axioma, mainwindow

app = init_axioma()
mainwindow.show()
app.exec_()
```

В процессе ввода кода VS Code будет отображать подсказки.



3.4.6 Разработка плагинов (модулей) для Аксиомы.

В папке /opt/axioma-gis5/bin/python_plugins/ находятся встроенные модули Аксиомы. Их можно использовать как пример при написании плагинов.

Чтобы начать разработку плагина, можно скопировать минимальный пример модуля ru_axioma_gis_axipy_example_plugin_minimal в папку проекта.

```

pythonProject # папка с проектом
├── ru_axioma_gis_axipy_example_plugin_minimal # модуль
│   ├── __init__.py # Точка входа модуля
│   └── manifest.ini # Описание модуля

```

Далее, нужно переименовать папку с модулем, например example_plugin_minimal (Название папки является уникальным идентификатором плагина). В файле manifest.ini нужно изменить отображаемое имя, чтобы найти плагин в списке модулей Аксиомы, например: name=Минимальный пользовательский плагин

Затем, нужно запустить Аксиому и добавить папку проекта в Дополнительные пути загрузки модулей. (Вкладка Основные | Настройки | Модули | Дополнительные модули | Настройки) Далее, нужно включить плагин в списке установленных плагинов. Теперь, при каждом запуске Аксиомы, плагин будет автоматически загружаться.

Подробнее о разработке плагинов можно прочитать в разделе [Разработка Плагинов \(Модулей\)](#).

Примечание: Windows 7. Последняя поддерживаемая версия PyCharm, доступная для скачивания: 2019.3. Скачать ее можно по ссылке: <https://www.jetbrains.com/pycharm/download/other.html>. Последняя поддерживаемая версия VS Code, доступная для скачивания: 1.70.2. Скачать ее можно по ссылке: https://code.visualstudio.com/updates/v1_70.

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.CoordSystem`. Объекты типа `axipy.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;

Полный список доступных функций создания систем координат содержится в документации к классу `axipy.CoordSystem`.

Примечание: Для выполнения примеров:

```
from axipy import CoordSystem, CoordTransformer
```

Список 1: Например

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

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Меркатора WGS84
'''
```

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

Список 2: Например

```
crs = CoordSystem.from_string('prj:Earth Projection 12, 62, "m", 0')
print(crs.name)
'''
>>> Робинсона NAD27
'''
```

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 отдельный класс `axipy.CoordTransformer`. Он позволяет преобразовывать координаты между двумя СК.

Список 3: Например

```
transformer = CoordTransformer('epsg:4326', 'epsg:26953')
coordinate = (55.76, 37.6)
result = transformer.transform(coordinate)
print(f"Point({result.x}, {result.y})")
'''
>>> Point(8513601.095442554, 9873107.576049749)
'''
```


Объекты данных

Разные типы данных обобщены одним абстрактным типом `axipy.DataObject`. Он образует иерархию объектов данных различного типа: таблица, растр, грид, чертеж, панорама, и так далее.

5.1 Таблицы

Для того, чтобы мы могли работать как с географической, так и с атрибутивной информацией, данные в ГИС Аксиома 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 ГИС Аксиома. Таблица `axipy.Table` является наследником класса объекта данных `axipy.DataObject`.

5.1.1 Открытие таблиц

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

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

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

```
print(table.provider)
```

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

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 используйте конкретный провайдер данных из коллекции провайдеров `axipy.ProviderManager`. Он содержит все методы, параметры и допустимые значения, для работы с конкретным типом объектов данных.

Если по какой-то причине использовать конкретный провайдер данных не удастся, то используется функция `axipy.ProviderManager.open()`, которая принимает словарь со всеми необходимыми параметрами.

Пример открытия таблицы из базы данных PostgreSQL:

Примечание: Исключительно в качестве примера. Намного проще напрямую использовать провайдер данных `axipy.PostgreDataProvider`.

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

Открытие источников с множеством таблиц

Для получения всех доступных таблиц одного источника используется функция `axipy.ProviderManager.read_contents()`:

```
contents = axipy.provider_manager.read_contents('../path/to/datadir/example.gpkg')
print(contents)
```

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

Для источников с единственной таблицей список будет содержать одно имя:

```
contents = axipy.provider_manager.read_contents({'src': '../path/to/datadir/worldcap.
↪tab'})
print(contents)
```

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

5.1.2 Схема таблицы

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

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

Схему таблицы можно получить, используя свойство `axipy.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]
```

Атрибуты схемы

Атрибут представлен типом `axipy.Attribute`. Его главные параметры - это имя `axipy.Attribute.name` и тип `axipy.Attribute.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
<code>string</code>	строка
<code>int</code>	целое число
<code>double</code>	вещественное число с плавающей запятой
<code>decimal</code>	вещественное число с фиксированной запятой
<code>bool</code>	логическое значение
<code>date</code>	дата
<code>time</code>	время
<code>datetime</code>	дата и время

Специальный атрибут Система Координат `axipy.Schema.coordsystem` содержит значение СК 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.Attribute` содержит вспомогательные функции `axipy.Attribute.string()` и другие для создания атрибутов; для создания схемы таблицы используется конструктор - `axipy.Schema`.

Например, так можно создать схему таблицы:

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

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

Чтение записей

Объект таблицы `axipy.Table` дает возможность работать с записями `axipy.Feature`. Метод `axipy.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.

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

или найден автоматически из расширения файла:

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

См.также:

`axipy.ProviderManager.tab`, `axipy.ProviderManager.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)

```

Иногда может потребоваться массовая вставка записей в таблицу. В случае, если это делать по одной записи, то после каждой вставки будут отрабатываться события на изменение данных, которые в свою очередь используются в инструментах и прочих GUI компонентах. Для того, чтобы это избежать предлагается два метода. Рассмотрим их на примере вставки 1000 записей в таблицу:

1. С предварительных занесением в список `list` и последующей единовременной вставкой:

```

table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()
features = []
for i in range(1, 1000):
    fpoint = Feature({}, geometry = point, style = pstyle )
    features.append(fpoint)
table.insert(features)
table.commit()

```

Но при использовании данного метода при большом количестве данных есть вероятность перерасхода памяти. Этого можно избежать, если воспользоваться вторым подходом.

2. Использование функции-генератора. При этом читаемые данные сразу же используются при вставке. Этот метод можно использовать, как пример, если необходимо зачитать данные из текстового файла с неподдерживаемым форматом.

```

table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()

def generate_features(): # функция-генератор
    for i in range(1, 1000):
        yield Feature(geometry = point, style = pstyle)

table.insert(generate_features())
table.commit()

```

Или же это можно записать в другом виде (результат аналогичный):

```

table = provider_manager.openfile('sample.tab')
point = Point(1,1)
pstyle = PointStyle()
generator = ( Feature(geometry = point, style = pstyle) for i in range(1, 1000) )
table.insert(generator)
table.commit()

```

5.1.4 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 = provider_manager.openfile('../path/to/datadir/world.tab')
schema = Schema(
    Attribute.string('Страна'),
    Attribute.integer('Население'),
)
copy_table = provider_manager.createfile('../path/to/edited_world.tab', schema)
orig_features = orig_table.items()

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

При редактировании таблицы так-же присутствует возможность более гибкого управления производимыми в таблице изменениями. Допустимо выполнение отката как назад (отмена последних изменений) `axipy.Table.undo()`, так и откат вперед (возврат после выполнения отката назад) `axipy.Table.redo()`.

```
table.insert(feature1)
table.insert(feature2)
if table.can_undo:
    table.undo()
```

В рассмотренном примере будет вставлена только feature1.

5.1.5 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.DataManager`. Запрос выполняется относительно всех таблиц, находящихся в этом каталоге, с помощью метода `axipy.DataManager.query()`.

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

Список поддерживаемых функций можно найти в руководстве пользователя ГИС Аксиома и в диалоге построения SQL-запросов самого приложения ГИС Аксиома.

Интерфейс Qt

Более низкоуровневый доступ к данным возможен через стандартный Qt интерфейс `PySide2.QtSql` и вспомогательный `axipy.sql`. Это позволяет:

- избежать лишних округлений и нормализации значений, так как нет схемы таблицы `axipy.Schema` и атрибутов `axipy.Attribute`;
- не конвертировать значения, так как нет приведения к `axipy.Feature`;
- выделять меньше ресурсов, так как результат читается по одной записи без создания объекта данных `axipy.Table`;
- проще интегрировать с другим Qt кодом.

Примечание: Доступ к колонкам с геометрией и стилем осуществляется через значения `axipy.sql.geometry_uid` и `axipy.sql.style_uid`.

Все открытые таблицы образуют базу данных, которую можно получить функцией `axipy.sql.get_database()`.

```
db = axipy.sql.get_database()
query = QSqlQuery(db)
query.exec_('SELECT * FROM world WHERE Население > 100000000')
while query.next():
    print(query.value(0))
```

Обработка ошибок

При выполнении запроса возвращается признак успешности выполнения. Если выполнение оказалось неуспешным, функция `PySide2.QtSql.QSqlQuery.lastError()` может показать последнюю ошибку. Стандартная обработка ошибок в `PySide2.QtSql` выглядит следующим образом:

```
q = QSqlQuery(database)
success = q.exec_(sql_text)
if not success:
    # Передаем ошибку выше
    raise RuntimeError(q.lastError().text())
```

5.2 Растры

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

Класс `axipy.Raster` представляет растровый объект данных.

Примечание: Для выполнения примеров:

```
from axipy import provider_manager, Layer, Map, view_manager, Raster, raster
```

5.2.1 Открытие растров, расположенных в файловой системе

Для открытия растровых файлов используйте функцию `axipy.ProviderManager.openfile()` объекта `axipy.provider_manager`.

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

5.2.2 Открытие из СУБД

Так-же поддерживается открытие данных из СУБД [PostgreSQL](#) и [Oracle](#). Данный функционал реализован через передачу строки в формате библиотеки [GDAL](#). Пример открытия растра из БД PostgreSQL и показа его на карте:

```
raster = provider_manager.gdal.open("PG:host=server_name dbname='test' user='postgres'  
↪ ' password='password' schema='public' table=table_name")  
raster_layer = Layer.create(raster)  
print(raster_layer)  
map = Map([ raster_layer ])  
mapview = view_manager.create_mapview(map)
```

Во избежании хранения строки соединения в открытом виде, некоторые ключи можно передать как дополнительные параметры:

```
raster = axipy.provider_manager.gdal.open("PG:host=server_name dbname='test' user=  
↪ 'postgres' schema='public' table='table_name'", open_data = {'PASSWORD': 'password'})
```

Пример открытия растра из БД Oracle:

```
raster = provider_manager.gdal.open("GeoRaster:user/password@XE,GDAL_RDT,1")
```

где `user/password@XE` - строка соединения с БД, `GDAL_RDT,1` - источник, который необходимо открыть. Подробнее см [gdalinfo](#).

5.2.3 Открытие данных, расположенных на WEB-ресурсах

Пример открытия тайлового сервера TMS. При этом передается шаблон URL:

```
raster = provider_manager.tms.open('https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}'  
↪ '.png')
```

Пример открытия WMTS:

```
raster = provider_manager.wmts.open('https://basemap.at/wmts/1.0.0/WMTSCapabilities.'  
↪ '.xml', 'geolandbasemap')
```


5.2.4 Регистрация растра

Растры по определению имеют пространственную привязку - координатную систему `axipy.Raster.coordsystem` и точки привязки `axipy.Raster.get_gcps()`. Таким образом:

Изображение + Пространственная привязка = Растр

Для того, чтобы «привязать» растр (задать изображению пространственную привязку), можно воспользоваться функцией `axipy.register()` модуля `axipy.raster`.

Например так можно привязать растр, используя эквивалентную матрицу преобразования (`PySide2.QtGui.QTransform` по умолчанию). Т.е. точка на изображении (x, y) в пикселях будет привязана к точке в пространстве (x, y) в единицах Системы Координат.

```
from PySide2.QtGui import QTransform

matrix = QTransform()
coordsystem = CoordSystem.from_units(Unit.m)
register(imagefile, matrix, coordsystem)
```

Чтобы привязать растр по-другому, можно передать другую матрицу трансформации (аффинное преобразование). Предварительно ее придется узнать или посчитать. Для расчета матрицы преобразования достаточно 3 точек привязки `axipy.GCP`. Обычно по углам изображения. Функция `axipy.register()` также может это сделать.

```
gcps = [
    GCP((0, 0), (0, 1000)),
    GCP((100, 0), (1000, 1000)),
    GCP((0, 100), (0, 0)),
]
coordsystem = CoordSystem.from_string('prj:NonEarth 0,7')
register(imagefile, gcps, coordsystem, True)
```

5.2.5 Трансформация растра

Операция трансформации `axipy.raster.transform()` растрового файла требуется для устранения или компенсации искажений, возникающих при создании растра. Требуется, когда аффинного преобразования недостаточно.

Список 1: Пример использования

```
coordsystem = CoordSystem.from_epsg(4326)
gcps = [
    GCP((0, 0), (0, 0)),
    GCP((200, 0), (30, 30)),
    GCP((200, 200), (60, 0)),
]
transform(rasterfile, outputfile, gcps, coordsystem)
```

См.также:

Руководство пользователя ГИС Аксиома раздел «Растровые изображения»

Провайдеры данных

За каждый тип данных отвечает провайдер данных `axipy.ProviderManager`. Провайдер владеет информацией о том, как представлять конкретный тип объектов данных и как ими манипулировать. Класс `axipy.ProviderManager` содержит все зарегистрированные провайдеры данных.

Каждый провайдер имеет свои особенности и возможности, но общие концепции, такие как открытие и создание, выделены в общий интерфейс `axipy.DataProvider`. В то же время конкретный провайдер может иметь дополнительные функции и параметры, свойственные только ему.

Для получения списка загруженных провайдеров есть функция `axipy.ProviderManager.loaded_providers()`.

Например:

```
provider_manager.loaded_providers()
```

```
{'CsvDataProvider': 'Файловый провайдер: Текст с разделителями',
 'DwgDgnFileProvider': 'Провайдер DWG и DGN',
 'GdalDataProvider': 'Растровый провайдер GDAL',
 'MifMidDataProvider': 'Провайдер данных MIF-MID',
 'OgrDataProvider': 'Векторный провайдер OGR',
 'ShapeDataProvider': 'Shapefile',
 'PgDataProvider': 'PostgreSQL',
 'OracleDataProvider': 'oracle',
 'MsSqlDataProvider': 'MS SQL Server',
 'RestDataProvider': 'ArcGIS REST',
 'SqliteDataProvider': 'Векторный провайдер sqlite',
 'TabDataProvider': 'MapInfo',
 'TmsDataProvider': 'Тайловые сервисы',
 'WfsDataProvider': 'Web Feature Service',
 'WmsDataProvider': 'Web Map Service',
 'WmtsDataProvider': 'Web Map Tile Service',
 'XlsDataProvider': 'Провайдер чтения файлов Excel'}
```

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 провайдеров данных определены функции задания источников данных `axipy.DataProvider.get_source()` с

дополнительными параметрами. Например, `axipy.CsvDataProvider.get_source()`, `axipy.PostgreDataProvider.get_source()` и другие.

Например:

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

Что эквивалентно:

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

Или:

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

См.также:

Подробнее в документации `axipy.ProviderManager`.

6.1 Открытие/Создание

Открытие и создание объектов данных `axipy.DataObject` выполняется провайдерами данных. Класс `axipy.ProviderManager` представляет коллекцию зарегистрированных провайдеров данных.

6.1.1 Открытие

Самый простой способ открыть объект данных из файла - использовать функцию `axipy.ProviderManager.openfile`. Функция сама найдет подходящий провайдер данных для открытия.

Совет: Это предпочтительный способ.

Список 1: Пример открытия

```
# filepath = 'path/to/file.tab'
table = provider_manager.openfile(filepath)
```

Примечание: При открытии данных они попадают в единый каталог `axipy.DataManager`.

При необходимости указать больше параметров для открытия, или если параметры по умолчанию не подходят, можно явно использовать провайдер данных. Необходимый провайдер данных можно получить из коллекции зарегистрированных провайдеров данных `axipy.ProviderManager`.

Список 2: Пример открытия конкретным провайдером

```
# csv_filepath = 'path/to/file.csv'
csv_table = provider_manager.csv.open(csv_filepath, delimiter='\\t')
```

Самый сложный способ - открытие через словарь `dict`. Здесь нужно заранее знать все параметры, их допустимые значения и идентификатор провайдера данных.

Список 3: Пример открытия из словаря

```
"""
Открывает csv файл через определение ``definition``.
Исключительно в качестве примера. Открывать csv проще провайдером.
"""
# csv_filepath = 'path/to/file.csv'
definition = {
    'src': csv_filepath,
    'delimiter': '\\t',
    'charset': 'utf8',
    'hasNamesRow': True,
    'provider': 'CsvDataProvider'
}
csv_table = provider_manager.open(definition)
```

При открытии таблиц из СУБД задаются как параметры подключения, так и требуемая таблица или текст запроса.

Список 4: Пример открытия данных из БД Postgres с указанием имени таблицы.

```
definition = provider_manager.postgre.get_source(
    host='192.168.0.2',
    db_name='test',
    user='test',
    password='pass',
    dataobject='world'
)
table = provider_manager.open(definition)
```

Список 5: Пример открытия данных из БД oracle с указанием текста SQL запроса.

```
definition = provider_manager.oracle.get_source(
    host='192.168.0.2',
    db_name='ORCL',
    user='test',
    password='pass',
    sql='select * from world'
)
table = provider_manager.open(definition)
```

Если источник содержит в себе несколько объектов данных, то этот список можно получить посредством метода `axipy.ProviderManager.read_contents()`:

```
# Запросим перечень таблиц
definition = {'src': 'sample.gpkg'}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print(provider_manager.read_contents(definition))
>>> ['table1', 'table2', 'table3']
```

Далее подставляем имя нужной таблицы и открываем объект данных.

```
# Откроем нужную таблицу
definition = {'src':'sample.gpkg', 'dataobject':'table1'}
table = provider_manager.open(definition)
```

6.1.2 Создание

Аналогично, самый простой способ создания файлов - с помощью функции `axiру.ProviderManager.createfile`:

Список 6: Пример создания

```
# filepath = 'path/to/file.tab'
schema = Schema(
    Attribute.string('country', 30),
    Attribute.string('capital', 30),
    coordsystem='prj:Earth Projection 1, 104'
)
table = provider_manager.createfile(filepath, schema)
```

Примечание: При создании объекта данных он автоматически открывается.

Для задания дополнительных параметров или использования специализированных возможностей провайдера данных, можно явно вызывать методы конкретного провайдера.

Список 7: Пример создания конкретным провайдером

```
schema = Schema(
    Attribute.string('country', 30),
    Attribute.string('capital', 30),
)
temp_table = provider_manager.shp.open_temporary(schema)
```

Если и этот способ не подходит для решения какой-то задачи, можно создавать объекты данных из словаря. Это самый сложный и непрактичный способ.

6.2 Импорт/Экспорт

6.2.1 Экспорт

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

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

Экспортировать можно отдельные записи, таблицы или целые источники данных.

Список 8: Пример экспорта таблицы в формат CSV

```
# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.csv.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True)
```

Выполнить экспорт можно так же следующим образом:

Список 9: Пример экспорта таблицы в формат MIF

```
# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.mif.get_destination(output_filepath, table_src.schema)
destination.export(table_src.items())
```

Если требуется создать таблицу и занести в нее некоторую информацию, но при этом формат данных не поддерживается на изменение, эту проблему можно решить, используя временную таблицу как буфер. Т.е. формируем данные в памяти, а потом производим экспорт этой временной таблицы в нужный нам формат.

Список 10: Пример формирования файла csv на базе временной таблицы

```
# Создаем временную таблицу
definition = {
    'src': '',
    'schema': attr.schema(
        attr.integer('id'),
        attr.string('name')
    )
}
table_src = provider_manager.create(definition)
# Добавляем в нее записи
table_src.insert(Feature({'id': '1', 'name': 'Name1'}))
table_src.insert(Feature({'id': '2', 'name': 'Name2'}))
# Создаем целевую таблицу и производим экспорт
destination = provider_manager.csv.get_destination('outfile.csv', Schema())
destination.export_from_table(table_src, copy_schema=True)
```

Пример преобразования в БД

Рассмотрим экспорт таблицы в базу данных на примере.

Для PostgreSQL:

```
# откроем исходный файл
src = provider_manager.openfile('points.tab')
# Обозначим дополнительные параметры при экспорте. В данном случае это явное
↳ пересоздание таблицы
# и сохранение результатов в лог файле. Данный параметр можно опустить.
exportParams = ExportParameters()
exportParams.logFile = 'export.log'
exportParams.dropTable = True
# Наименование таблицы в БД
newTableName = 'points_db'
# Определяем источник куда будем производить экспорт
dest = provider_manager.postgre.get_destination(host='db_server_addr',
                                                db_name='test',
                                                dataobject=newTableName,
                                                user='user',
                                                password='pass',
                                                export_params = exportParams,
                                                schema = src.schema)
# Непосредственно производим экспорт
dest.export_from_table(src)
```

Для MSSQL Server:

```
dest = provider_manager.mssql.get_destination(host='db_server_addr5',
                                              db_name='test',
                                              dataobject=newTableName,
                                              user='user',
                                              password='pass',
                                              export_params = exportParams,
                                              schema = src.schema)
dest.export_from_table(src)
```


Пример преобразования из DWG и Панорамы

Если требуется экспортировать данные из таких источников, как ГИС Панорама или данных AutoCAD, то это можно сделать двумя путями:

- Вся информация по всем слоям отправляется в один файл
- Производится разбивка по слоям, или же выборочная выборка нужных слоев.

Рассмотрим первый вариант для файла AutoCAD.

```
# Откроем исходный файл
definition = {'src':'input_file.dwg'}
data_object = provider_manager.open(definition)
out_folder = '/mnt/hdd/export_folder'
file_name = os.path.join(out_folder, 'output_file.tab')
destination = provider_manager.tab.get_destination(file_name, data_object.schema)
destination.export(data_object.items())
```

По аналогии для данных ГИС Панорама.

```
# Откроем исходный файл
data_object = provider_manager.openfile('Podolsk.map')
out_folder = '/mnt/hdd/export_folder'
file_name = os.path.join(out_folder, 'output_file.tab')
destination = provider_manager.tab.get_destination(file_name, data_object.schema)
destination.export(data_object.items())
```

Рассмотрим второй вариант. Все слои экспортируем как отдельные файлы.

```
# Откроем исходный файл
definition = {'src':'input_file.dwg'}
data_object = provider_manager.open(definition)
out_folder = '/mnt/hdd/export_folder'
# Запросим список слоев и каждый слой экспортируем в отдельный файл
for layer_name in data_object.layers:
    file_name = os.path.join(out_folder, '%s'.join([layer_name, 'tab']))
    destination = provider_manager.tab.get_destination(file_name, data_object.schema)
    destination.export(data_object.items(layer_name))
```

Список слоев можно взять как у уже открытого объекта посредством метода `axipy.RasteredTable.layers()`, так и у менеджера провайдеров, передав ему определение (как при открытии источника) `axipy.ProviderManager.read_contents()`

Экспортируем один слой из файла формата ГИС Панорама в файл TAB.

```
data_object_src = provider_manager.openfile('Podolsk.map')
out_file_name = 'output_file.tab'
destination = provider_manager.tab.get_destination(out_file_name, data_object_src.
↪ schema)
destination.export(data_object_src.items('ГИДРОГРАФИЯ'))
```

Пример преобразования в DWG и Панораму

Следующим примером произведем конвертацию из формата TAB в формат AutoCAD DWG. При этом последовательно обработаем два файла и поместим данные из них на соответствующие слои.

```
out_file_name = 'outfile.dwg'
# Целевая Система Координат
cs = CoordSystem.from_prj('0,7')
# Экспортируем первый файл на слой в AutoCAD 'world'
table_src = provider_manager.openfile('world.tab')
destination = provider_manager.dwg.get_destination(out_file_name, table_src.schema,
↪ 'world', coordsystem = cs)
destination.export(table_src.items())
# Экспортируем второй файл на слой в AutoCAD 'SubjectRF' при этом указываем open_mode
table_src = provider_manager.openfile('SubjectRF.TAB')
destination = provider_manager.dwg.get_destination(out_file_name, table_src.schema,
↪ 'SubjectRF', coordsystem = cs,
↪ open_mode = OpenMode.Append)
destination.export(table_src.items())
```

Экспортируем таблицу в формате TAB в файл Панорамы SIT. Для этого дополнительно нам необходимо указать классификатор и наименование ключевого поля

```
# Файл с классификатором
rsc = 'Podolsk/Topo100t.rsc'
# Исходный файл
table_src = provider_manager.openfile('ГИДРОГРАФИЯ.tab')
# Наименование целевого файла
out_filename = 'sit_outfile.sit'
# Определяем целевой файл
destination = provider_manager.panorama.get_destination(out_filename, rsc, table_src.
↪ schema, 'ObjectKey')
# Производим экспорт
destination.export(table_src.items())
# Экспортируем второй слой
table_src = provider_manager.openfile('РАСТИТЕЛЬНОСТЬ.tab')
destination = provider_manager.panorama.get_destination(out_filename, rsc, table_src.
↪ schema, 'ObjectKey', open_mode = OpenMode.Append)
destination.export(table_src.items())
```

Преобразование в формат Sqlite и GPKG

Пример экспорта в GPKG последовательно 2-х таблиц в один выходной файл:

```
# Определяем имя выходного файл и тип формата (GPKG)
file_path_out = "out_sqlite_GPKG.gpkg"
f = SqliteGeometryFormat.GPKG

# Экспортируем первый файл
t1 = axipy.provider_manager.openfile('world.tab')
destination = axipy.provider_manager.sqlite.get_destination(file_path_out, t1.name,
↪ t1.schema, geometry_format=f)
destination.export(t1.items())
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Экспортируем второй файл
t2 = axipy.provider_manager.openfile('worldcap.tab')
destination = axipy.provider_manager.sqlite.get_destination(file_path_out, t2.name,
    ↪ t2.schema, geometry_format=f)
destination.export(t2.items())
```

Для экспорта в формат sqlite необходимо изменить строку, задающую выходной формат:

```
f = SqliteGeometryFormat.FGF
```

Преобразование с использованием библиотеки OGR

Для этого воспользуемся провайдером `axipy.OgrDataProvider`.

Пример экспорта из файла формата MapInfo в Shapefile с созданием пространственного индекса:

```
t = axipy.provider_manager.openfile('in/states.tab')
data = {"lco":{"SPATIAL_INDEX":"YES"}}
destination = provider_manager.shp.get_destination('out/out_shp.shp', t.schema,
    ↪ create_data=data)
destination.export(t.items())
```

Экспорт в источник, который задан строкой соединения. Рассмотрим пример экспорта из файла формата MapInfo в БД PostgreSQL:

```
t = axipy.provider_manager.openfile('in/states.tab')
conn_str="PG:host='localhost' dbname='test' user='test' password='pass' "
data = {"lco":{"GEOMETRY_NAME":"AXIOMA_GEOMETRY", "FID":"AXIOMA_ID", "OVERWRITE":"YES",
    ↪ ""}}
destination = provider_manager.ogr.get_destination(conn_str, 'states', t.schema,
    ↪ create_data=data)
destination.export(t.items())
```

Запись двух слоев в sqlite:

```
tw = axipy.provider_manager.openfile('in/world.tab')
twc = axipy.provider_manager.openfile('in/worldcap.tab')
destination = provider_manager.ogr.get_destination('out_sqlite.sqlite', 'world', tw.
    ↪ schema)
destination.export(tw.items())
destination = provider_manager.ogr.get_destination('out_sqlite.sqlite', 'worldcap',
    ↪ twc.schema, open_mode = OpenMode.Append)
destination.export(twc.items())
```

В KML:

```
destination = provider_manager.ogr.get_destination('out_kml.kml', 'world', t.schema)
destination.export(t.items())
```

В GML:

```
destination = provider_manager.ogr.get_destination('out_gml.gml', 'world', t.schema)
destination.export(t.items())
```

Экспорт из PostgreSQL в GPKG средствами OGR:

```
src = provider_manager.postgre.get_source(host='localhost',
                                         db_name='test',
                                         dataobject='world',
                                         user='test',
                                         password='pass')

t=axipy.provider_manager.open(src)
destination = provider_manager.ogr.get_destination('out_sqlite.sqlite', 'worldcap', t.
↪ schema, open_mode = OpenMode.Create)
destination.export(t.items())
```

6.2.2 Импорт

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

Список 11: Пример экспорта источника

```
source = provider_manager.tab.get_source(input_tab_file)
destination.export_from(source)
```

См.также:

Чтобы узнать, какие типы поддерживают импорт и экспорт, обратитесь к описанию конкретных провайдеров данных `axipy.ProviderManager`.

6.2.3 Изменение структуры таблицы

Если необходимо изменить структуру таблицы, в частности добавить новое поле, удалить существующее или же его расширить. Эту процедуру можно выполнить через экспорт. Рассмотрим на примере.

Откроем исходную таблицу

```
# открываем исходный файл
src = provider_manager.openfile(src_filepath)
```

Далее, возьмем схему исходной таблицы, сделаем ее копию и произведем все необходимые с ней изменения. Т.е. добавим новое поле в конкретную позицию, удалим ненужное и расширим длину существующего.

```
# Целевая схема как копия исходной
dest_schema = src.schema
# вставка нового поля
dest_schema.insert(2, Attribute.string('Новое поле', 30))
# Удаление поля по имени
attr_to_del = dest_schema.by_name('Capital')
dest_schema.remove(attr_to_del)
# Увеличение длины поля
idx_edit = dest_schema.index_by_name('Страна')
dest_schema[idx_edit] = Attribute.string('Страна', 100)
```

Следующим шагом создаем выходную таблицу на базе полученной схемы и производим непосредственно экспорт.

```
# Создаем таблицу
dest = provider_manager.tab.get_destination(dest_filepath, dest_schema)
# Экспортируем данные
dest.export_from_table(src)
```

Если необходимо создать таблицу в памяти на базе измененной структуры, то эту задачу можно выполнить следующим образом:

```
# Создаем таблицу в памяти на базе измененной схемы
memory_table = provider_manager.create({'schema': dest_schema})
# Производим вставку данных
memory_table.insert(src.items())
```


Запись `axipy.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.

7.1 Атрибуты

Доступ к 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:
    ...
```

7.2 Геометрический атрибут

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

Или можно использовать специальное наименование `axipy.Feature.GEOMETRY_ATTR`, представляющее имя геометрического атрибута:

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

Проверка существования `axipy.Feature.has_geometry()`:

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

7.3 Стиль для геометрического атрибута

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

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

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

7.4 Идентификаторы записей

Записи имеют свойство `axipy.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.

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.

8.1 Типы

ГИС 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.

8.1.1 Точка

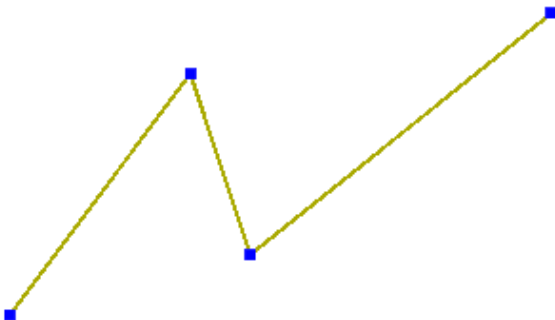
Точка представлена классом `axipy.Point`. Для нее характерно отсутствие таких параметров, как площади и линейных размеров. Создадим точку с координатами (10, 10) в СК Широта/Долгота. С целью визуального восприятия, результат представим в виде WKT строки (`axipy.Geometry.to_wkt()`).

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

8.1.2 Полилиния

Представлена классом `axipy.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.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.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.Geometry.to_wkt()`:

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

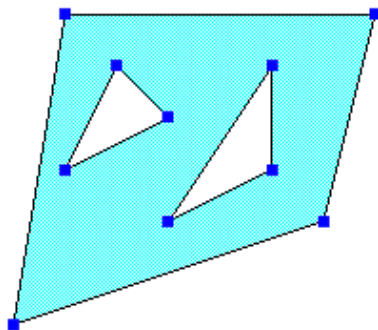
8.1.3 Полигон

Представлен классом `axipy.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.to_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.to_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.Polygon.holes`. Обновим значение третьей точки для второй дырки.

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

8.1.4 Смешанная коллекция

Представлена классом `axipy.GeometryCollection`. 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 = GeometryCollection()
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.to_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.to_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.to_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.to_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.

8.1.5 Коллекция точек

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

```
mpoint = MultiPoint()
mpoint.append(11, 11)
mpoint.append((12, 12))
mpoint.append(point)
print(mpoint.to_wkt())
'''
>>> MULTIPOINT (11 11, 12 12, 10 10)
'''
```

8.1.6 Коллекция полилиний

Представлена классом `axipy.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.to_wkt())
'''
>>> MULTILINESTRING ((11 12, 13 14, 15 16), (21 22, 23 24, 25 26))
'''
```

8.1.7 Коллекция полигонов

Представлена классом `axipy.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(polygon) # Добавим ранее созданный с дыркой
print(mpoly.to_wkt())
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> 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.

8.1.8 Линия

Представлена классом `axipy.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)
```

8.1.9 Прямоугольник

Представлен классом `axipy.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.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)  
...
```

8.1.10 Скругленный прямоугольник

Представлен классом `axipy.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.Rect` и радиусы скругления:

```
rrectangle = RoundRectangle([0, 0, 40, 20], 0.2, 0.2)  
rrectangle.xRadius = 0.3  
print(repr(rrectangle))  
...  
>>> RoundRectangle xmin=0.0 ymin=0.0 xmax=40.0 ymax=20.0 xradius=0.3 yradius=0.2  
...
```

8.1.11 Эллипс

Представлен классом `axipy.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)
'''
```

8.1.12 Дуга

Представлена классом `axipy.Arc`. Линейный объект. Задается в конструкторе через `axipy.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
'''
```

8.1.13 Текст

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

```
rv = view_manager.create_reportview()
rect = Rect(8, 6, 11, 7)
text = Text("Пример", rect, view=rv)
geomItem = GeometryReportItem()
geomItem.geometry = text
geomItem.style = Style.for_geometry(text)
rv.report.items.add(geomItem)
```

Рассмотрим общие свойства геометрии.

8.2 Свойства геометрии

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.get_area()}')
print(f'Периметр: {polygon.get_perimeter()}')
print(f'Ограничивающий прямоугольник: {polygon.bounds}')
'''
>>> Название: Полигон
>>> Площадь: 100.0
>>> Периметр: 40.0
>>> Ограничивающий прямоугольник: (0.0 0.0) (10.0 10.0)
'''

```

8.3 Сериализация

ГИС 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.Geometry.from_wkt()` для получения объекта из WKT и метод `axipy.Geometry.to_wkt()` для получения WKT представления полученного объекта:

```

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)

```

Аналогично сделаем для WKB. При этом используются, соответственно, `axipy.Geometry.from_wkb()` и `axipy.Geometry.to_wkb()`:

```

wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00$@'
pnt = Geometry.from_wkb(wkb)

```

8.4 Преобразования

Для перепроецирование в другую СК используется метод `axipy.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.to_wkt()))
'''

```

(continues on next page)

(продолжение с предыдущей страницы)

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

Более простые преобразования типа сдвига `axipy.Geometry.shift()`, масштабирования `axipy.Geometry.scale()` и поворот `axipy.Geometry.rotate()` 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.to_wkt()))
polygon_scale = polygon.scale(5, 5)
# Отмасштабируем координаты относительно центра
print('polygon scale: {}'.format(polygon_scale.to_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.to_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.Geometry.affine_transform()` с указанием матрицы преобразований `QTransform`

8.5 Пространственные операции

8.5.1 Нормализация объекта

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

Создадим заведомо неправильный полигон и попробуем его исправить:

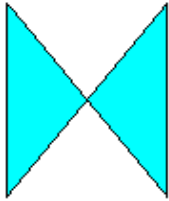
```

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.to_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.



8.5.2 Клонирование объекта

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

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

8.5.3 Логические операции

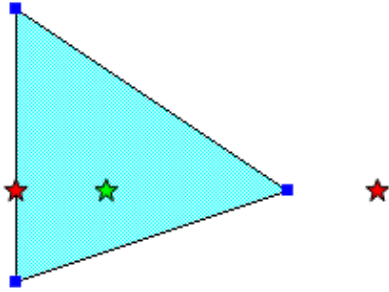
Пространственные отношения, возвращающие логический `True` или `False`:

Точное совпадение геометрий производится посредством `axipy.Geometry.equals()`, если же необходимо произвести приблизительное сравнение, то используем метод `axipy.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
'''
```

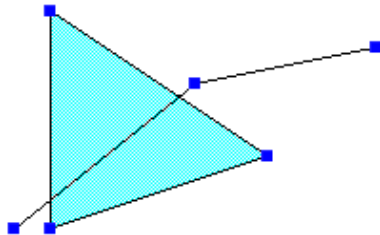
Проверка на попадание `axipy.Geometry.contains()`

```
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.Geometry.crosses()`

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
sl = LineString((0, 1), (5, 5), (10, 6))
print(poly1.crosses(sl))
...
>>> True
...
```



Проверка на отсутствие соприкосновений `axipy.Geometry.disjoint()`

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

Проверка пересечений объектов `axipy.Geometry.intersects()`

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

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

Проверка касания `axipy.Geometry.touches()`

```
point3 = Point(1, 5)
print('Точка на грани:', poly1.touches(point3))
...
>>> True
...
```

Функция, обратная contains `axipy.Geometry.within()`

```
print('Точка внутри:', Point(2, 5).within(poly1))
...
>>> True
...
```

Охват одной геометрии другой `axipy.Geometry.covers()`

8.5.4 Отношения DE-9IM

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

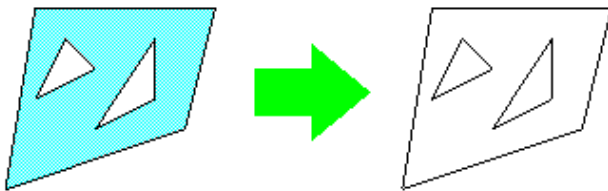
```
print('relate:', poly1.relate(Point(10, 10)))
...
relate: FF2FF10F2
...
```

8.5.5 Операции над объектами

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

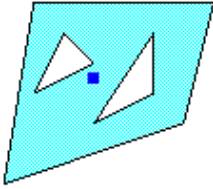
Получение границ геометрии в виде полилинии `axipy.Geometry.boundary()`

```
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.to_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))
...
```



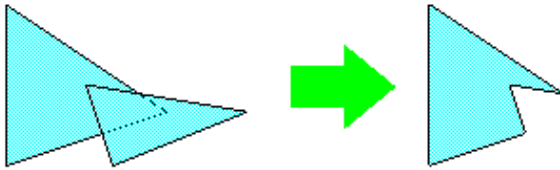
Центроид объекта `axipy.Geometry.centroid()`

```
centroid = poly.centroid()
print(centroid.to_wkt())
...
>>> POINT (4 4.5)
...
```



Вычитание объектов `axipy.Geometry.difference()`

```
poly1 = Polygon((1, 1), (1, 7), (7, 3))
poly2 = Polygon((5, 1), (4, 4), (10, 3))
print(poly1.difference(poly2).to_wkt())
'''
>>> POLYGON ((1 1, 1 7, 6 3.67, 4 4, 4.6 2.2, 1 1))
'''
```



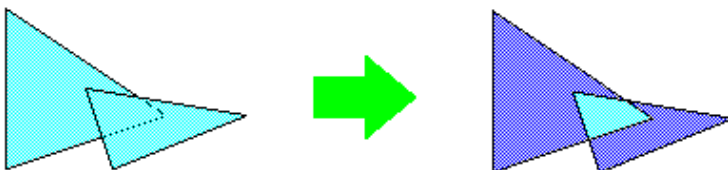
Пересечение объектов `axipy.Geometry.intersection()`

```
print(poly1.intersection(poly2).to_wkt())
'''
>>> POLYGON ((6 3.67, 7 3, 4.6 2.2, 4 4, 6 3.67))
'''
```



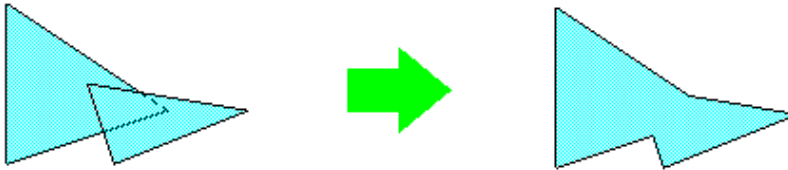
Обратное пересечение объектов `axipy.Geometry.symmetric_difference()`

```
print(poly1.symmetric_difference(poly2).to_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)))
'''
```



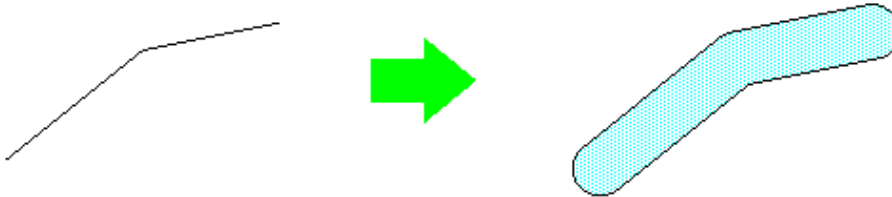
Объединение объектов `axipy.Geometry.union()`

```
print(poly1.union(poly2).to_wkt())  
...  
>>> POLYGON ((1 1, 1 7, 6 3.67, 10 3, 5 1, 4.6 2.2, 1 1))  
...
```



Построение буфера `axipy.Geometry.buffer()`

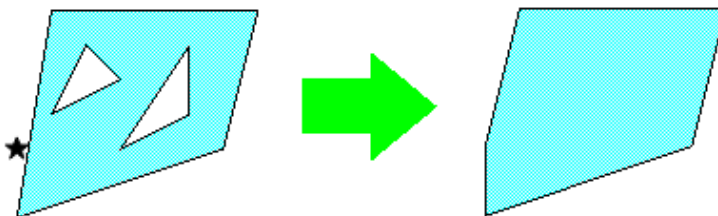
```
sl = LineString((0, 1), (5, 5), (10, 6))  
buf = sl.buffer(1)
```



Границы объекта `axipy.Geometry.convex_hull()`

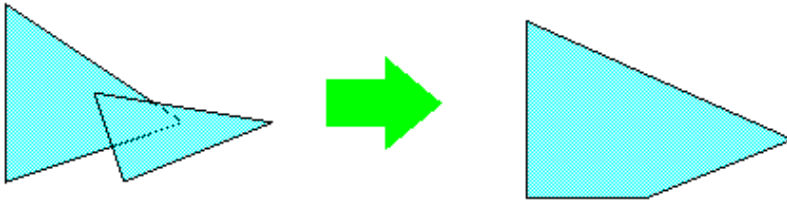
Рассмотрим на примере коллекции:

```
coll = GeometryCollection()  
coll.append(polygon)  
coll.append(Point(1, 5))  
print(coll.convex_hull().to_wkt())  
...  
POLYGON ((1 1, 1 5, 2 7, 8 7, 7 3, 1 1))  
...
```



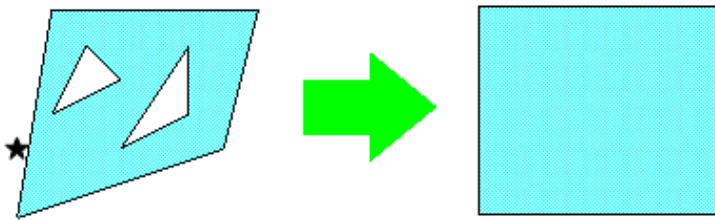
Пример с внутренними углами:

```
print(poly1.union(poly2).convex_hull().to_wkt())  
...  
POLYGON ((1 1, 1 7, 10 3, 5 1, 1 1))  
...
```



Прямоугольные границы объекта `axipy.Geometry.envelope()`

```
print(coll.envelope().to_wkt())
'''
POLYGON ((1 1, 8 1, 8 7, 1 7, 1 1))
'''
```



8.5.6 Конвертация объектов

Список 1: Пример конвертации полигона в полилинию.

```
polygon = Polygon((0, 0), (0, 10), (10, 10))
print(polygon.to_linestring().to_wkt())
'''
>>> LINESTRING (0 0, 0 10, 10 10, 0 0)
'''
```

Список 2: Пример конвертации полилинии в полигон.

```
ls = LineString((0, 0), (0, 10), (10, 10))
print(ls.to_polygon().to_wkt())
'''
>>> POLYGON ((0 0, 0 10, 10 10, 0 0))
'''
```


Стиль представляет собой описательную структуру, которая в свою очередь используется при оформлении геометрического объекта `Geometry` при его отрисовке. Стиль представлен базовым классом `axipy.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': Schema(
        Attribute.string('id', 60),
        coordsystem='prj:1, 104'
    )
}
table = provider_manager.create(definition)
```

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_manager.create_mapview(m)
```

В результате получим точку на карте:



Так же доступно создание из строки формата MapBasic. Для этого используется метод `axipy.Style.from_mapinfo()`. Если же для существующего стиля необходимо получить строку MapBasic, то используется метод `axipy.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.

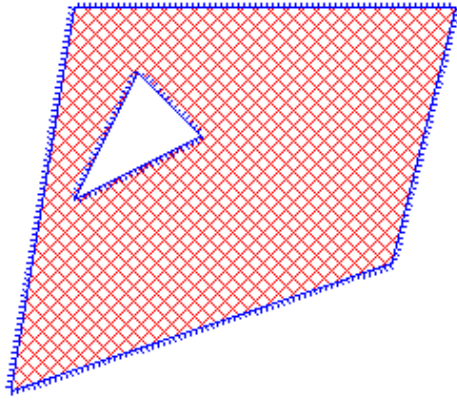
```
from PySide2.QtCore import Qt

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(
    geometry=polygon,
    style=polystyle
)
```

(continues on next page)

(продолжение с предыдущей страницы)

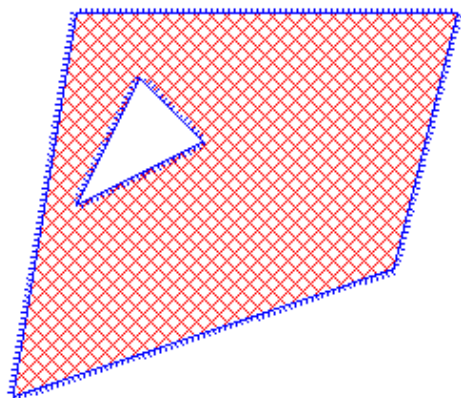
```
table.insert([fpolygon])
'''
>>> Pen (1, 48, 255) Brush (8, 16711680, 0)
'''
```



Для сложных разнородных объектов применяются стили, которые внутри себя содержат другие стили для каждого типа используемых объектов. Для этого используется класс `axipy.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:



10.1 Слой

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

```
from axipy.render import Layer  
  
layer = Layer.create(table)
```

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

Свойства подписей

Настройки автоматического подписывания определяются классом `axipy.Label` свойством `axipy.VectorLayer.label`.

Список 1: Пример использования

```
world = generate_layer_for_geometry_table()  
# Зададим в качестве формулы метки атрибут "Страна" и запретим перекрытие меток друг  
#    <-->другом:  
world.label.text = "Страна"  
world.label.placementPolicy = LabelOverlap.DisallowOverlap  
# Задание стиля оформления слоя  
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
```

Настройки автоматического подписывания `axipy.Label` повторяют соответствующие настройки в интерфейсе ГИС Аксиома диалога «Свойства слоя» > «Подписи».

См.также:

Подробнее в разделе «Подписывание» руководства пользователя для ГИС Аксиома.

10.2 Карта

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

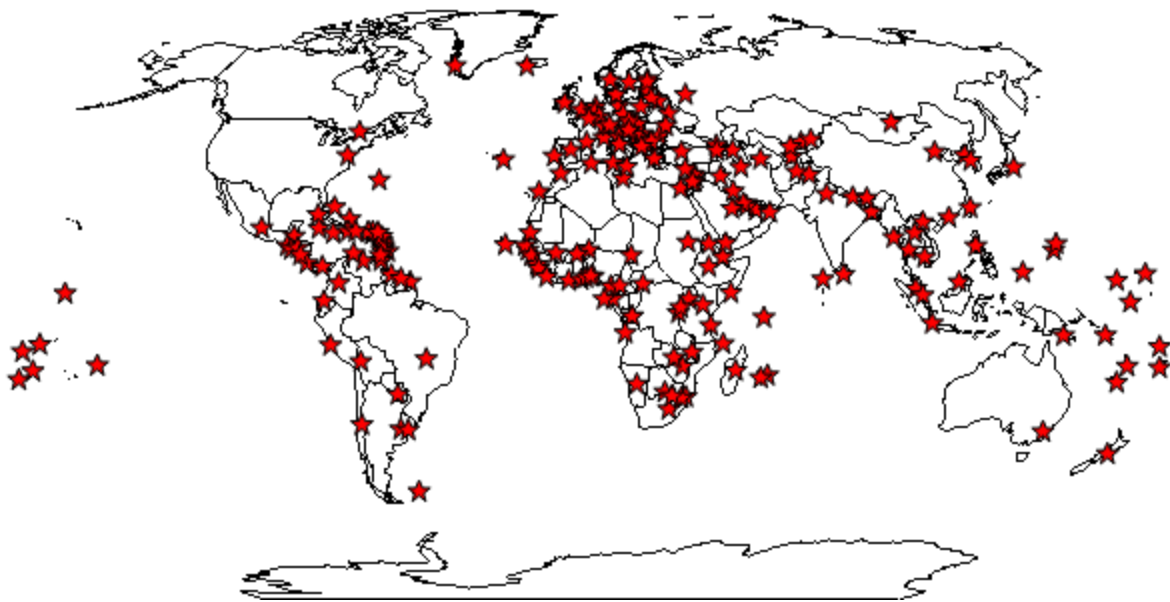
Создадим карту с двумя слоями. Передадим в карту список таблиц - из них автоматически создадутся слои с параметрами по умолчанию.

```
import axipy

world = axipy.provider_manager.openfile('../path/to/datadir/example.gpkg', dataobject=
    ↪ 'world')
capital = axipy.provider_manager.openfile('../path/to/datadir/worldcap.tab')
map_ = axipy.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:

```
print(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.VectorLayer.label`, и для обоих слоев переопределить `стиль оформления`, сделав его однообразным `axipy.VectorLayer.overrideStyle`. Заметим, что перечень доступных слоев карты доступен через свойство `axipy.Map.layers`. Т.е. помимо передачи перечня слоев в конструктор карты, также возможно управление этим списком позже.

```
from axipy import Label, Style

lay_capital = map_.layers[0]
lay_capital.label.text = 'Столица'
lay_capital.label.placementPolicy = Label.DisallowOverlap
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 прикладу по управлінню шарами в кінці видалимо шар зі столицями (самий верхній):

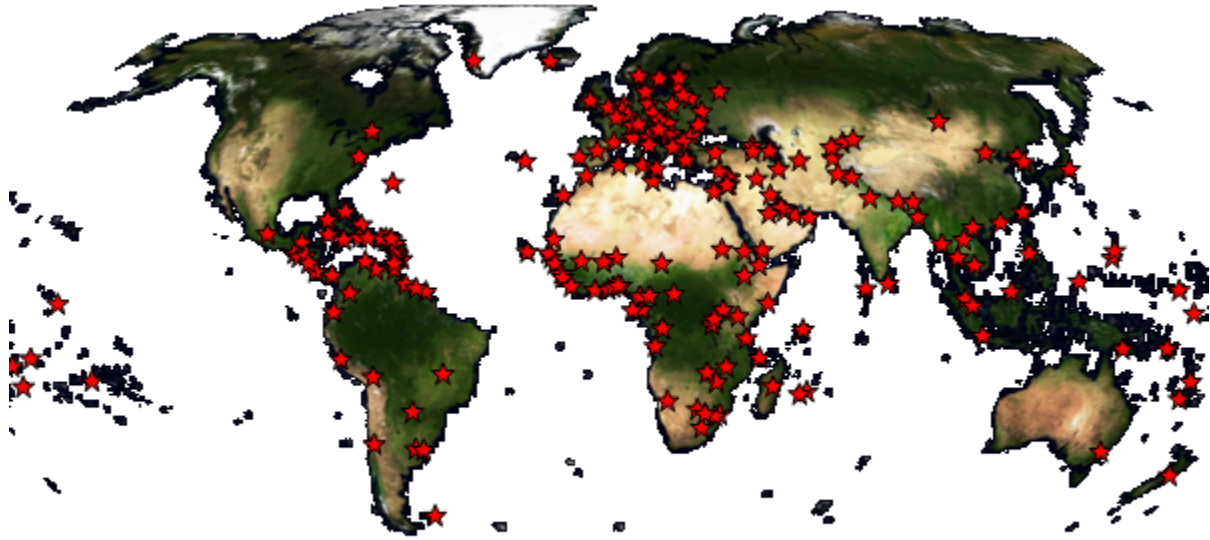
```
map_.layers.remove(0)
```

Складемо карту на базі растра як підложки і встановимо прозорчим колір фону.

```
from PySide2.QtGui import QColor
from axipy import provider_manager, Layer, Map

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

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



10.2.1 Окно редактирования карты

Окно карты `axipy.Мар`, созданное программно, можно экспортировать в виде растра или же поместить в отчет. Если же стоит задача проведения некоторых манипуляций с картой, таких как редактирование геометрии, то для проведения подобных манипуляций ее необходимо поместить в окно редактирования `axipy.MapView`. Для создания этого окна необходимо использовать метод `axipy.ViewManager.create_mapview()`. Данный метод доступен через сервис `view_manager`. Рассмотрим на примере:

Список 2: Пример использования.

```
# Откроем таблицу, создадим на ее базе слой и добавим в карту
table_world = provider_manager.openfile(filepath)
world = Layer.create(table_world)
map = Map([world])
# Для полученной карты создадим окно просмотра
mapview = view_manager.create_mapview(map)
```

При желании непосредственно в окно карты можно поместить элементы управления. Рассмотрим пример добавления ползунка в левый верхний угол окна карты, который изменяет прозрачность верхнего слоя карты:

```
from PySide2.QtWidgets import QSlider, QFrame, QLabel, QVBoxLayout
from PySide2.QtCore import Qt

def change_opacity(v):
    mv = view_manager.active
    if mv and len(mv.map.layers):
        mv.map.layers[0].opacity = 100 - v

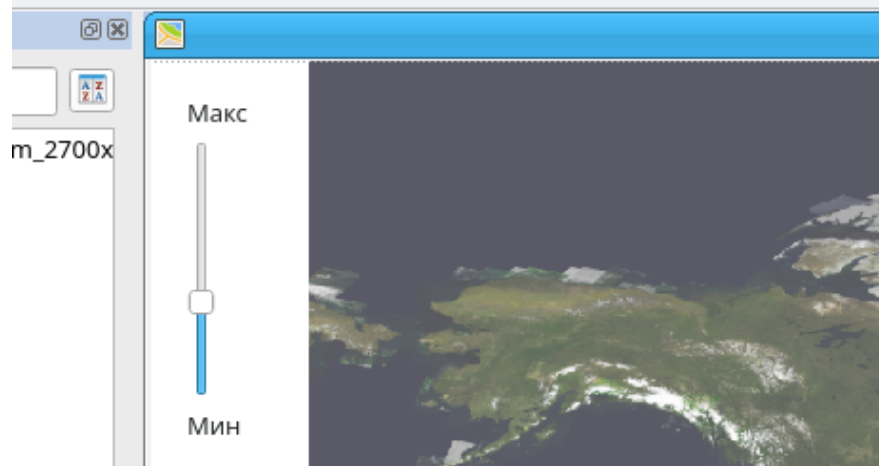
mv = view_manager.active
if mv is not None:
    frame = QFrame(mv.widget)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
layout = QVBoxLayout()
slider = QSlider(Qt.Vertical, frame)
slider.setRange(0, 100)
slider.valueChanged.connect(change_opacity)
layout.addWidget(QLabel("Макс", frame))
layout.addWidget(slider)
layout.addWidget(QLabel("Мин", frame))
frame.setGeometry(10, 10, 50, 300)
frame.setLayout(layout)
frame.show()
```

Результат будет выглядеть примерно так:



10.3 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.VectorLayer` есть возможность формирования и отрисовки тематических слоев. Т.е. применить оформление на базе атрибутивной информации.

Тематические слои добавляются как дочерние к их базовому слою.

```
from axipy import RangeThematicLayer

world = map_.layers[0]
thematic = RangeThematicLayer('Население')
world.thematic.add(thematic)
```

Поддерживаются следующие виды тематических слоев:

- `RangeThematicLayer` - Интервалы
- `PieThematicLayer` - Круговые диаграммы
- `BarThematicLayer` - Столбчатые диаграммы
- `SymbolThematicLayer` - Знаки

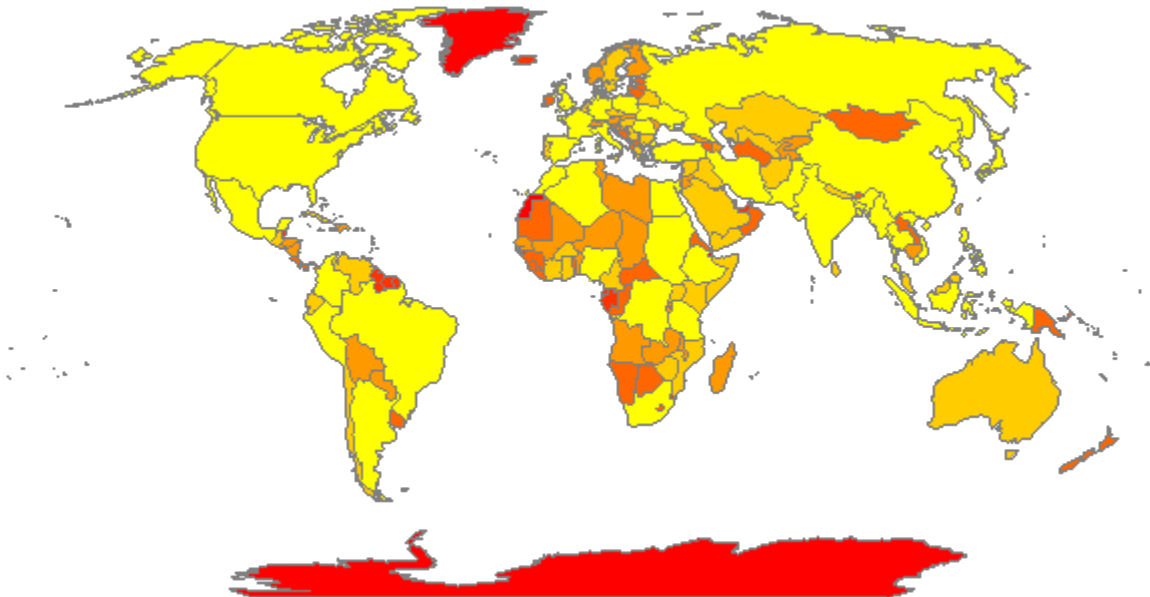
- `IndividualThematicLayer` - Индивидуальные значения
- `DensityThematicLayer` - Плотность точек

Для более удобного распределения по цветам используются различного рода алгоритмы. Выбор того или иного алгоритма обусловлен исходными требованиями к составлению тематики. Эти алгоритмы сгруппированы в базовом интерфейсе `axipy.ReallocateThematicColor` и могут быть использованы в наследниках. Поддерживаются следующие виды распределения:

- По двум заданным крайним цветам `axipy.ReallocateThematicColor.assign_two_colors()`.
- По двум заданным крайним цветам и цвету разрыва `axipy.ReallocateThematicColor.assign_two_colors()`.
- По спектру `axipy.ReallocateThematicColor.assign_rainbow()`.
- Градация серого `axipy.ReallocateThematicColor.assign_gray()`.
- Монотонная заливка разной яркости `axipy.ReallocateThematicColor.assign_monotone()`.

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.

```
table_world = provider_manager.openfile('world.tab')
world = Layer.create(table_world)
rangel = RangeThematicLayer("Население")
rangel.ranges = 6
rangel.splitType = RangeThematicLayer.EQUAL_COUNT
rangel.assign_two_colors(Qt.yellow, Qt.red)
world.thematic.add(rangel)
```



Поменяем стиль оформления для первого интервала:

```
rangel.set_style(0, PolygonStyle(45, Qt.blue))
```

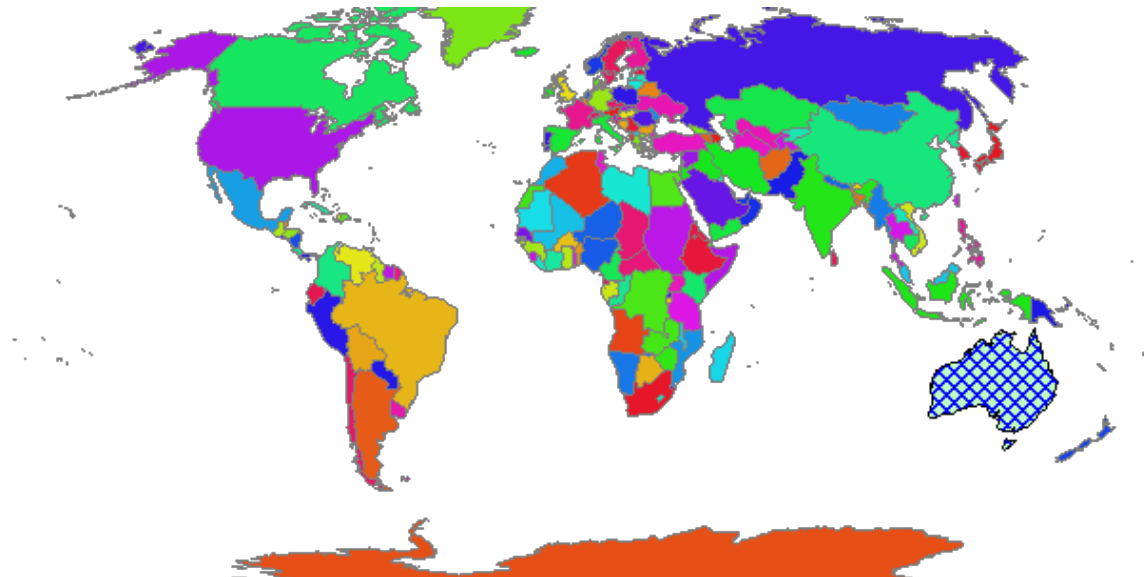
Так же есть возможность ручного переопределения значений разбивки по интервалам. Т.е. задание минимального и максимального значений требуемого интервала. Переопределим верхнее значение для первого интервала:

```
iv = range1.get_interval_value(0)
print('Old values:', iv)
range1.set_interval_value(0, (iv[0], 100000.0))
print('New values:', range1.get_interval_value(0))
```

```
>>> Old values: (0.0, 66687.0)
>>> New values: (0.0, 100000.0)
```

Создадим тематику с отдельными значениями по полю „Страна“. Распределение по цветам случайное:

```
individual = IndividualThematicLayer('Страна')
individual.assign_rainbow()
world.thematic.add(individual)
individual.set_style(0, PolygonStyle(45, Qt.blue))
```



Изменим цвет интервала по индексу 1 на желтый.

```
s = individual.get_style(1)
s.polygon.fill.color = Qt.yellow
individual.set_style(1, s)
```

Необходимую тематику слоя можно получить по ее индексу `axipy.Layer.thematic()`:

```
range1 = world.thematic[0]
```

Если необходимо просмотреть все тематики слоя:

```
for t in world.thematic:
    print('thematic:', t.title)
```

```
>>> thematic: Интервалы
>>> thematic: Значения
```

10.4 Легенда

Для вывода условных обозначений используется легенда. Легенда – таблица, содержащая образцы условных обозначений и письменных пояснений к ним. В ГИС Аксиома 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 PySide2.QtCore import Qt
from PySide2.QtGui import QImage, QPainter
from axipy import Legend, Context

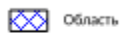
legend_world = Legend(lay_world)
legend_world.position = (10, 10)

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

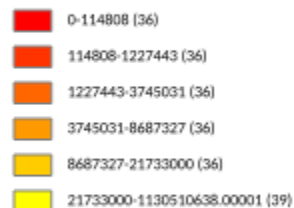
image = QImage(500, 200, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter_legend = QPainter(image)
context_legend = Context(painter_legend)

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

Легенда world



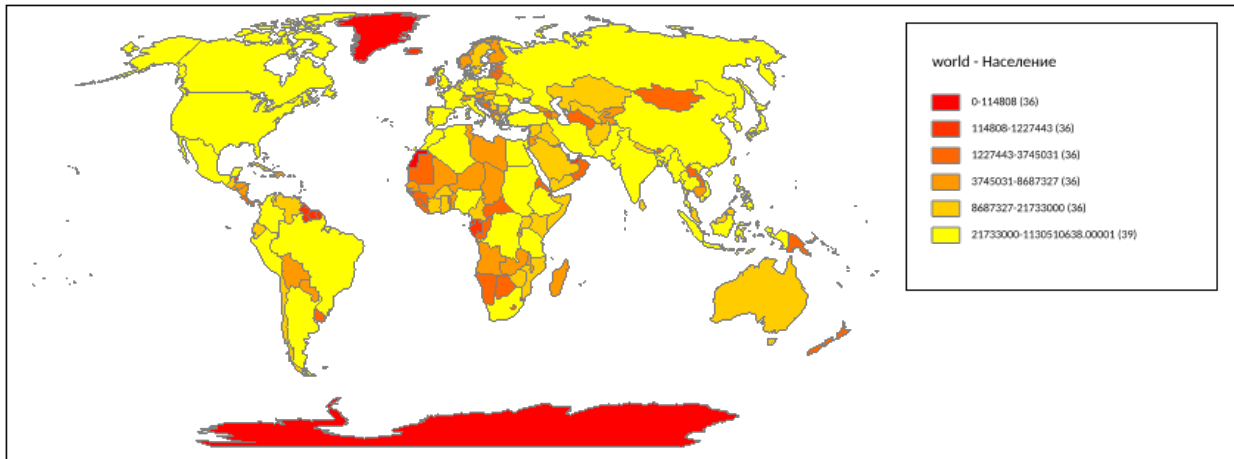
world - Население



Создадим легенду, на этот раз сразу переведа 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)
```



Доступ к элементам легенды производится через свойство `axipy.Legend.items`.

```
for it in legend_thematic.items:
    print(it.title, it.visible, it.style.to_mapinfo())
```

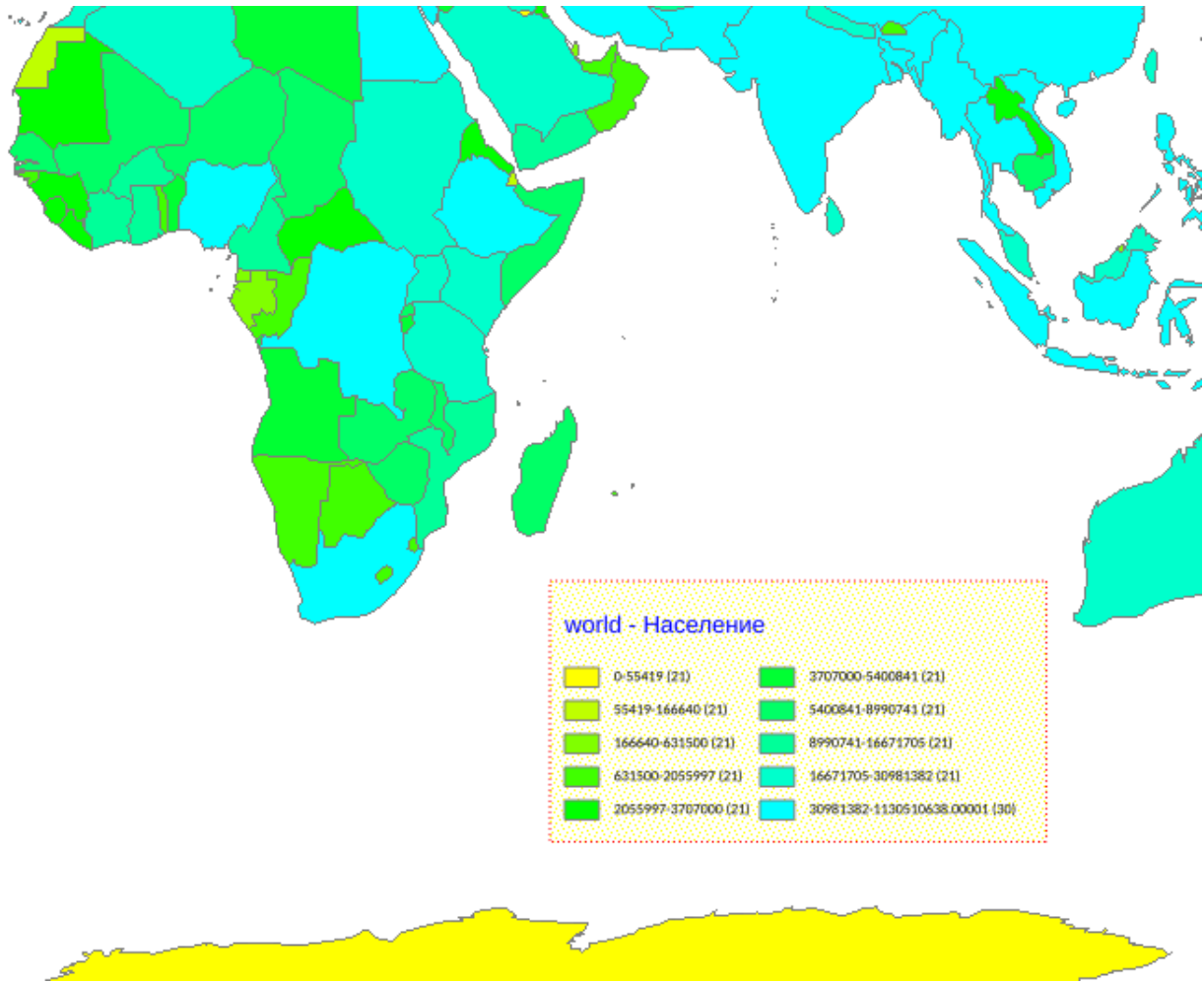
```
>>> 0-55419 True Pen (1, 2, 0) Brush (45, 255)
>>> 166640-631500 True Pen (1, 2, 8421504) Brush (2, 8453888)
>>> 631500-2055997 True Pen (1, 2, 8421504) Brush (2, 4259584)
```

Если требуется изменить какой-то элемент, к примеру сделать его скрытым или изменить описание, то в данном случае необходимо сначала запросить требуемый элемент, изменить его характеристики и обновить на модифицированный. Прямое изменение не поддерживается.

```
item = legend.items[0]
item.title = 'Описание'
legend.items[0] = item
```

Если легенду необходимо как-то выделить на общем фоне или она с ним сливается, то можно указать стиль рамки и заливки заднего фона:

```
legend.border_style = LineStyle(3, Qt.red)
legend.fill_style = PolygonStyle(49, Qt.yellow)
```



10.4.1 Окно легенды для карты

Для просмотра и редактирования легенды `axipy.Legend` для карты необходимо использовать метод `axipy.ViewManager.create_legendview()`, передав в него как параметр окно ранее созданной карты:

Список 3: Пример использования.

```
map = Map([world])
mapview = view_manager.create_mapview(map)
legendView = view_manager.create_legendview(mapview)
```

При этом будут помещены все доступные для просмотра легенды слоев карты `map_view`. Легенды окна можно посмотреть следующим образом:

Список 4: Пример использования.

```
for legend in legendView.legends:
    print(legend.caption)
...
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> World
...
```

10.5 Отчет

Для вывода информации на печать предусмотрено создание отчетов `axipy.Report`. 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 import Report

report = Report(printer)
```

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

```
geometryReportItem = GeometryReportItem()
geometryReportItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geometryReportItem.style = PolygonStyle(45, Qt.red)
report.items.add(geometryReportItem)
```

Аналогично добавим карту.

```
table = provider_manager.openfile('world.tab')
world = Layer.create(table)
map = Map([world])
mapReportItem = MapReportItem(Rect(10, 110, 200, 210), map)
mapReportItem.scale = 200000000
report.items.add(mapReportItem)
```

Контекст для печати по подобию рассмотренному контексту для карты.

```
from PySide2.QtGui import QPainter

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

Производим печать.

```
report.draw(context)
```

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

10.5.1 Окно редактирования отчета

Если необходимости в визуальном редактировании отчета `axipy.Report` нет, а требуется только программно создать макет отчета и распечатать его на принтере или сохранить как PDF, то достаточно создать `axipy.Report` и работать с ним. В противном случае отчет, по аналогии с картой для визуального редактирования его так же помещают в окно редактирования `axipy.ReportView`. В качестве примера создадим окно отчета и поместим в него геометрию и карту. Стоит заметить, что карту так-же можно использовать у уже открытого окна карты `axipy.MapView.map()`. Координаты элементов задаются в единицах измерения отчета `axipy.Report.unit`. В нашем случае это миллиметры.

Список 5: Пример использования.

```
from PySide2.QtCore import Qt
rv = view_manager.create_reportview()

# Добавим полигон
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
rv.report.items.add(geomItem)

# Добавим направляющую
rv.x_guidelines.append(20)

# Текущий масштаб просмотра
rv.view_scale = 33

# Включение режима привязки координат
rv.snap_mode = True

# Размер ячейки сетки
rv.mesh_size = 20

# Откроем и добавим карту
table = provider_manager.openfile(filepath)
world = Layer.create(table)
map = Map([world])
mapItem = MapReportItem(Rect(10, 100, 200, 200), map)
mapItem.scale = 200000000
rv.report.items.add(mapItem)

# Поменяем стиль у первого объекта
rv.report.items[0].style = PolygonStyle(45, Qt.blue)
```

11.1 Создание кнопки

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

```
from axipy import ActionButton, Position, tr

button = ActionButton("Простое действие", on_click=lambda: print("triggered"))
position = Position(tr("Основные"), tr("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.

11.2 Доступность кнопки

В приложении ГИС Аксиома многие кнопки и инструменты меняют свойство доступности в зависимости от текущего состояния. Например, если кнопка производит действие над выбранным объектом, то логично сделать эту кнопку доступной только при наличии выбранных объектов. Чтобы проще задавать подобное поведение для своих кнопок, предлагается использовать `axipy.Observer` и место их создания и получения `axipy.ObserverManager`.

Так, чтобы сделать кнопку активной только при наличии выборки, ее можно создать следующим образом, передав параметр `enable_on` функции `axipy.menubar.create_button()`:

```
from axipy import menubar, ObserverManager

button = menubar.create_button(
    "Имя кнопки",
    on_click=lambda: print("on_click"),
    enable_on=ObserverManager.Selection
)
```

Идентификаторы наблюдателей по умолчанию перечислены в `axipy.ObserverManager`.

Для инструментов идентификатор рекомендуется задавать в самом классе инструмента, переопределив параметр `axipy.MapTool.enable_on`.

Возможно добавление своих наблюдателей, в том числе их комбинация с уже существующими `axipy.ObserverManager.create()`.

Чтобы получить текущее значение наблюдателя, используется свойство `axipy.Observer.value`.

Создание виджетов

- Программное наполнение диалога
- Использование файла ресурсов *.ui. Прямая загрузка.
- Использование наследования на базе файла ресурсов *.ui

Для построения пользовательского интерфейса библиотека Qt, поставляемая в рамках ГИС Аксиома, предоставляет большой набор инструментария.

Для примера рассмотрим построение простейшего диалога. Целью данного примера является краткое описание того инструментария, который можно использовать для создания пользовательских форм и диалогов.

Для того, чтобы наш диалог обладал некоторыми специфичными свойствами, унаследуем его от стандартного базового класса `PySide2.QtWidgets.QDialog` и переопределим его поведение в соответствии с нашими потребностями. Первоначальный код будет выглядеть примерно так:

```
from PySide2.QtWidgets import QDialog
from axipy import view_manager

class MyDialog(QDialog):
    pass

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)
dialog.exec()
```

Здесь мы создали пользовательский класс диалога и активировали его. Отметим, что свойство `view_manager.global_parent`, переданное в конструктор рассматривается как объект-владелец данного диалога. Если данное свойство не проставить, то диалог в панели задач будет показан как отдельное приложение.

Тестирование можно производить в рамках самой ГИС Аксиомы. Для этого необходимо открыть редактор. Кнопка запуска находится на панели «Консоль Python». Если она отсутствует в интерфейсе, то для включения необходимо в выпадающем меню кнопки «Панели» включить пункт «Консоль Python». Сохраним данный файл в файловой системе под именем `mydialog.py`.

Далее, чтобы разместить на данном диалоге элементы управления можно пойти двумя путями:

- Создать эти элементы программно
- Использовать для этого файл ресурсов *.ui

Второй вариант более понятен и удобен, но мы вкратце рассмотрим оба подхода.

12.1 Программное наполнение диалога

Создадим простой диалог, на который программно добавим кнопку и поле ввода. По нажатию на кнопку в консоль будет выведен текст элемента ввода.

```
from PySide2.QtWidgets import QDialog, QPushButton, QLineEdit
from axipy import view_manager

class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        self.button = QPushButton(self)
        self.button.setText("Кнопка")
        self.button.move(150, 50)
        # Реакция на нажатие кнопки
        self.button.clicked.connect(self.button_clicked)
        self.edit = QLineEdit(self)
        self.edit.setText("Текст")
        self.edit.move(10, 50)

    def button_clicked(self):
        print(self.edit.text())

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)

dialog.exec()
```

В данном примере элементы располагаются с явным указанием их места. Это не совсем удобно и в зависимости от размеров элементов управления на конкретной системе могут накладываться друг на друга. Или при изменении размеров самой формы могут вообще исчезать. Для решения этой проблемы обычно используются классы динамического размещения элементов на форме. В нашем случае это наследники класса `PySide2.QtWidgets.QLayout`. Модифицируем пример с использованием класса `PySide2.QtWidgets.QGridLayout`.

```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QDialog, QPushButton, QLineEdit, QGridLayout
from axipy import view_manager

class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        # создаем layout
```

(continues on next page)

(продолжение с предыдущей страницы)

```

layout = QGridLayout()
self.button = QPushButton()
self.button.setText("Кнопка")
self.button.clicked.connect(self.button_clicked)
# Добавляем кнопку
layout.addWidget(self.button, 0, 0, Qt.AlignTop)
self.edit = QLineEdit()
self.edit.setText("Текст")
# Добавляем элемент ввода
layout.addWidget(self.edit, 0, 1, Qt.AlignTop)
# Назначаем layout для диалога
self.setLayout(layout)

def button_clicked(self):
    print(self.edit.text())

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)

dialog.exec()

```

Теперь наши элементы при изменении размеров диалога будут пропорционально менять свои размеры, прижимаясь к верхнему краю.

Если элементов на форме много, то такой способ размещения элементов достаточно трудоемок. Рассмотрим альтернативный способ размещения с использованием дизайнера [Qt Designer](#).

12.2 Использование файла ресурсов *.ui. Прямая загрузка.

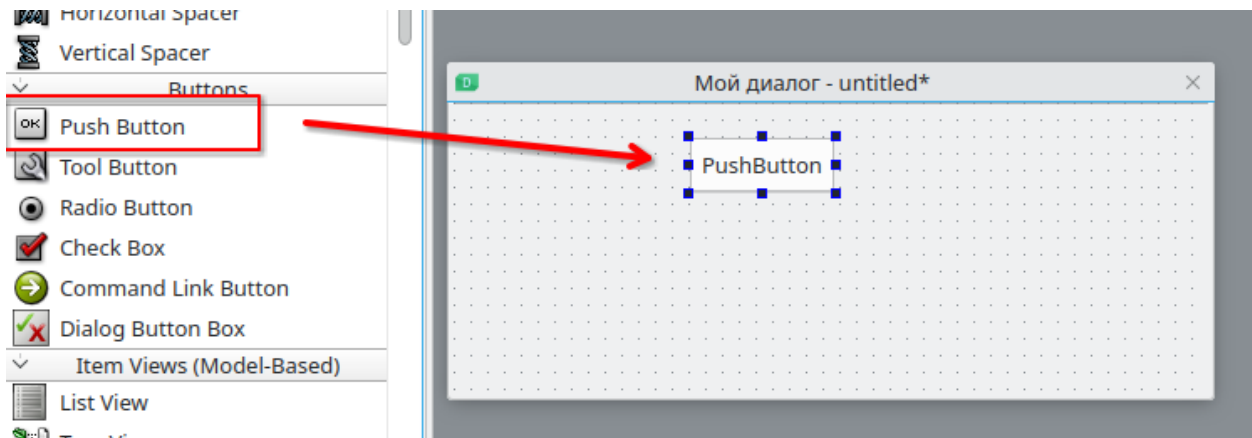
Для формирования UI представления формы нам понадобится инструмент [Qt Designer](#). Он поставляется совместно со средствами разработки Qt, но его также можно установить отдельно как пакет `python qt5 applications`. Подробнее о инсталляции пакетов см в разделе [Без интернета. Ручная установка пакетов](#).. Т.е. в конечном итоге в зависимости от окружения нам нужно выполнить команду:

```
python3 -m pip install --user qt5_applications
```

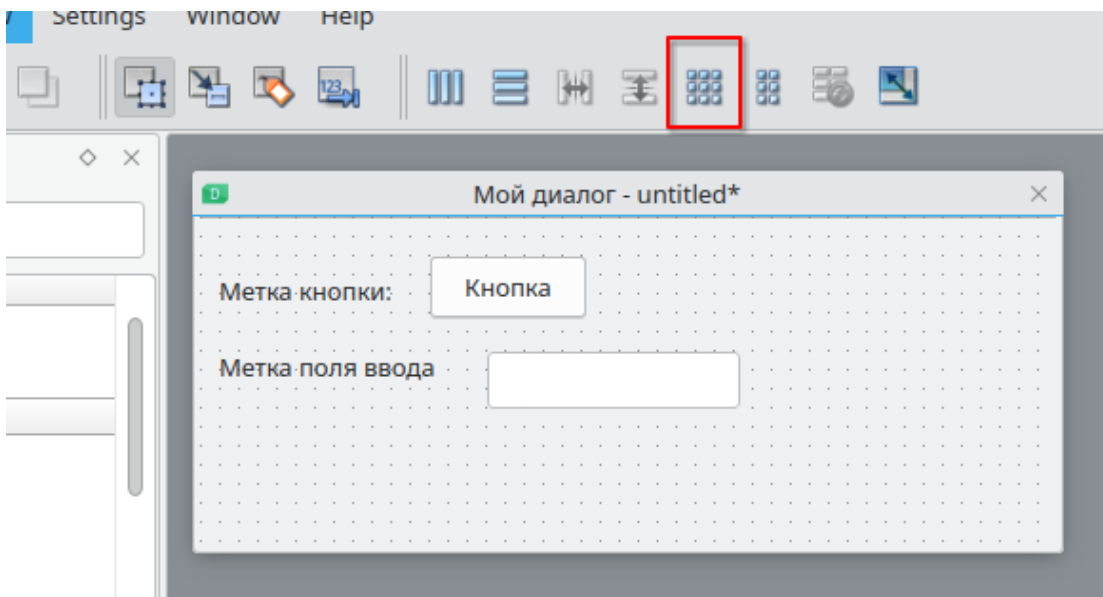
После успешной инсталляции в каталог [site-packages](#) из файлового менеджера переходим в каталог `site-packages/qt5_applications/Qt/bin` и запускаем на выполнение файл `designer`.

Как альтернативный и более простой вариант, можно воспользоваться возможностью установки и запуска `QtDesigner` посредством соответствующего модуля. Для этого на вкладке «Основные» необходимо нажать кнопку «Модули». Далее, на вкладке «Дополнительные модули» следует нажать «Загрузить список модулей». Из полученного списка выбираем «Qt Designer» и нажимаем «Установить». На панели «Основные» должна появиться кнопка «QtDesigner».

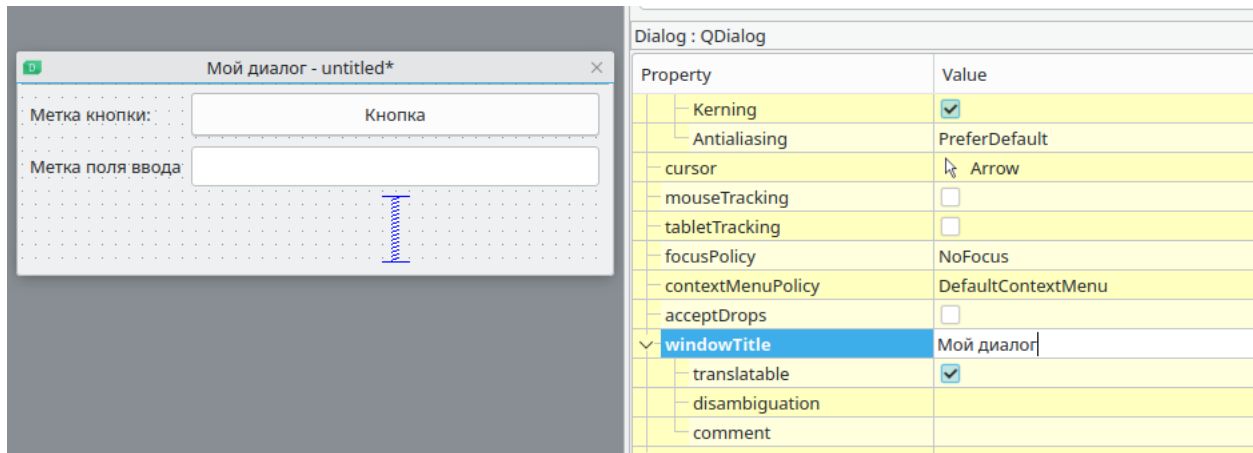
Запускаем инструмент. Выбираем тип диалога или формы. В нашем случае это `Dialog without buttons`. Далее, помещаем на него посредством перетаскивания мышью кнопку `PySide2.QtWidgets.QPushButton` и элемент ввода `PySide2.QtWidgets.QLineEdit`.



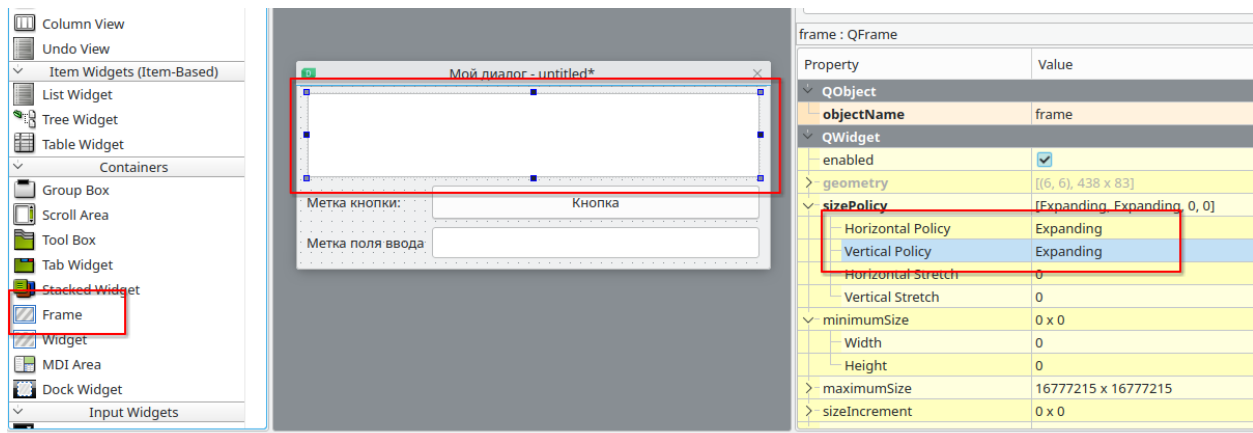
Слева от этих элементов поместить метки, перетащив элемент `PySide2.QtWidgets.QLabel`. Меняем у кнопки ее текст на «Кнопка» (двойным щелчком на элементах) и для поля ввода заносим текст «Текст». Также можно поменять названия элементов на `button` и `edit` (чтобы они совпадали с предыдущими примерами). Для этого после выделения соответствующего элемента, в таблице свойств возле свойства `objectName` установим значение `button`. Выберем диалог и установим свойство `windowTitle` равным «Мой диалог».



Для того, чтобы при изменении размеров формы расположение элементов на ней не уезжало, необходимо применить динамическое выравнивание (Layout). Далее, для нашей формы применим динамическое размещение элементов, выбрав на панели его тип. Для этого выберем мышью основную форму и нажмем кнопку на панели `Layout in a Grid`. Элементы займут все пространство и при изменении размеров окна, их размер пропорционально будет изменяться. Чтобы элементы прижать кверху, перетянем по аналогии с кнопкой и полем ввода элемент `Vertical Spacer` и отпустим его над нижней частью (но внутри) окна. После этого наши элементы должны с середины переместиться наверх.



Если мы хотим, чтобы при изменении размеров окна изменялись размеры одного из элементов, необходимо сделать следующее: Выделим и удалим ранее добавленный объект Vertical Spacer. После этого находим и перетягиваем объект `PySide2.QtWidgets.QFrame`. Кладем его по аналогии с удаленным Vertical Spacer, но помещаем его сверху на форму над всеми ранее добавленными элементами. Если он занял не все пространство, его необходимо растянуть мышью. После для свойств Horizontal и Vertical Policy устанавливаем значение Expanding.



Сохраняем файл под именем `mydialog.ui` и кладем его рядом с файлом `mydialog.py`.

Далее, нам необходимо этот файл загрузить в диалог. Измененный пример с аналогичным функционалом будет выглядеть следующим образом:

```
import os

from PySide2.QtUiTools import QUiLoader
from PySide2.QtWidgets import QDialog
from axipy import view_manager

class MyDialog(QDialog):
    def __init__(self, parent):
        super().__init__(parent)
        # Путь файла в файловой системе
        ui_file = os.path.join(os.path.dirname(__file__), "mydialog.ui")
        # Загружаем файл ui
```

(continues on next page)

(продолжение с предыдущей страницы)

```

self.ui = QUiLoader().load(ui_file, parent)
self.ui.button.clicked.connect(self.button_clicked)

def button_clicked(self):
    print(self.ui.edit.text())

# Перенаправим метод show на self.ui
def show(self):
    return self.ui.show()

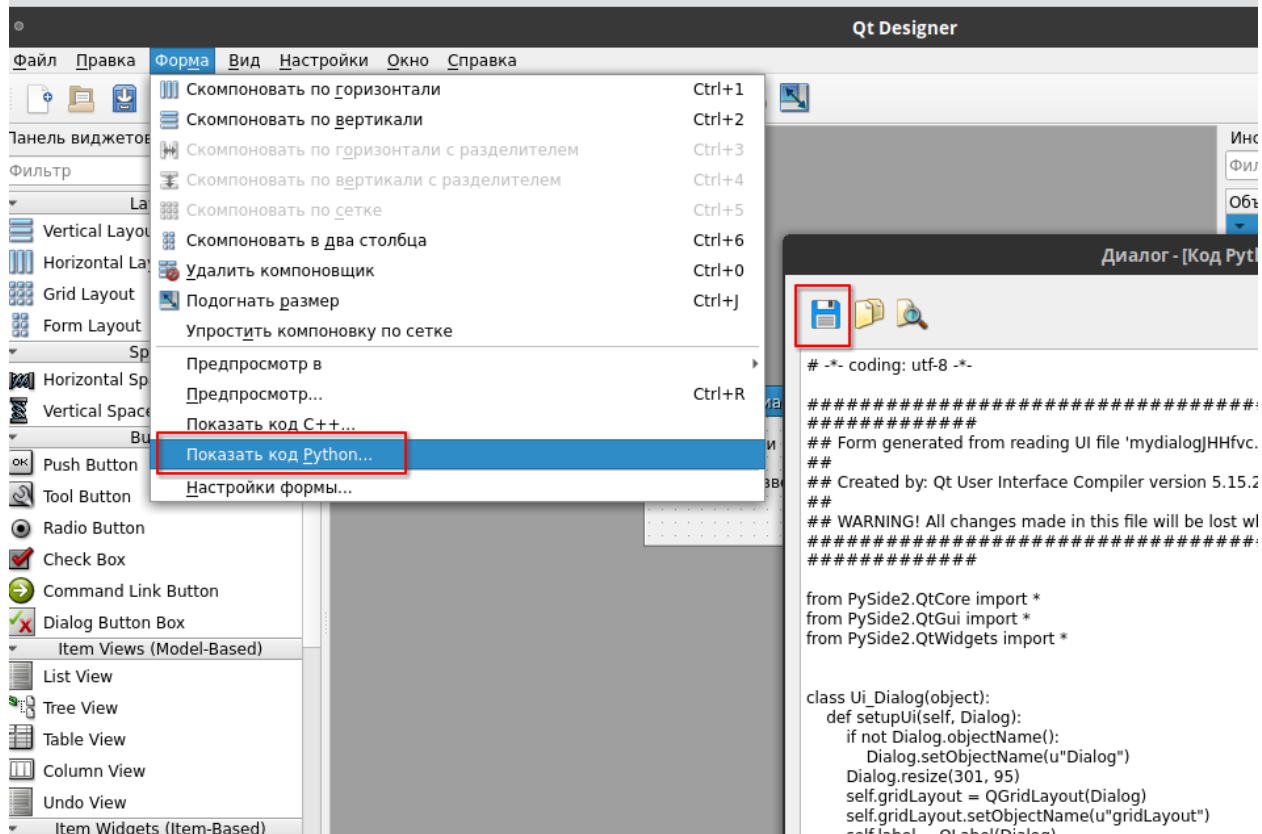
# Показ диалога как немодальное окно. Для модального используем метод exec
dialog = MyDialog(view_manager.global_parent)
dialog.resize(500, 300)
dialog.exec()

```

Стоит заметить, что к элементам теперь необходимо обращаться не через `self`, а через `self.ui`. В примерах поставки ГИС Аксиома есть подобное решение с реализацией в виде модуля `ru_axioma_gis_axiru_example_dialog`.

12.3 Использование наследования на базе файла ресурсов *.ui

Наряду с прямой загрузкой файла *.ui как ресурса можно использовать метод наследования классов. Для этого необходимо создать файл класса python *.py, из файла *.ui. В QtDesigner открываем файл из предыдущего раздела `mydialog.ui`. В меню выбираем Форма/Показать код python.... В появившейся форме нажимаем кнопку Сохранить и указываем имя `ui_mydialog.py`.



В итоге мы получили класс `Ui_Dialog`, который будем использовать как базовый для нашего диалога. Вторым базовым классом будет `PySide2.QtWidgets.QDialog`. Код будет выглядеть следующим образом:

```
# Импортируем сгенерированный класс
from .ui_mydialog import Ui_Dialog
from PySide2.QtWidgets import QDialog

class MyDialog(QDialog, Ui_Dialog):

    def __init__(self, parent=None):
        super().__init__(parent)
        # Загружаем объекты из сгенерированного Ui_Dialog
        self.setupUi(self)
        self.button.clicked.connect(self.button_clicked)

    def button_clicked(self):
        print(self.edit.text())

dlg = MyDialog(view_manager.global_parent)
dlg.exec()
```

После любого изменения файла `mydialog.ui` необходимо пересоздать файл `ui_mydialog.py`. Также, необходимо отметить, что вносить изменения в файл `ui_mydialog.py` не рекомендуется, т.к. после очередного изменения с последующим пересозданием, все правки будут утеряны.

Создание инструментов

13.1 Передача параметров в инструменты

Дополнительные параметры могут быть переданы прямо в конструктор инструмента `axipy.MapTool` при создании кнопки `ToolButton`. Для этого необходимо «обернуть» вызов конструктора в функцию, захватив (capture) все необходимые параметры. Например, сравните:

Список 1: Инструмент без параметров

```
button = ToolButton('Мой инструмент', MyTool)
```

Список 2: Инструмент с захватом параметров

```
button = ToolButton('Мой инструмент', lambda: MyTool(config, params))
```

13.2 Панель активного инструмента

Если при работе с инструментом пользователю регулярно нужно взаимодействовать с различными графическими элементами / настройками то для этого можно воспользоваться готовым решением из `axipy.ActiveToolPanel`. Это обертка вокруг стандартной панели `PySide2.QtWidgets.QDockWidget` предоставляющая дополнительные возможности:

1. Можно задать наблюдателя который будет автоматически следить за доступностью панели.
2. Предопределенное место для панели активного инструмента.
3. Готовые компоненты с кнопками управления для упрощения добавления пользовательского графического элемента.

При написании плагинов текущий экземпляр `axipy.ActiveToolPanel` можно получить из метода `axipy.AxiomaInterface.active_tool_panel()`. Взаимодействие с панелью активного инструмента идет через обработчик который можно создать

через один из методов `axipy.ActiveToolPanel.make_acceptable()` или `axipy.ActiveToolPanel.make_custom()`. При работе с панелью активного инструмента через обработчик `axipy.AxipyAcceptableActiveToolHandler` созданный через `axipy.ActiveToolPanel.make_acceptable()` то на панели активного инструмента по умолчанию расположены кнопки «Применить» и «Отмена». При нажатии на них будут посланы соответствующие сигналы `axipy.AxipyAcceptableActiveToolHandler.accepted` и `axipy.AxipyAcceptableActiveToolHandler.rejected`. Если нужно настроить кнопки управления по своему усмотрению то можно воспользоваться обработчиком `axipy.AxipyCustomActiveToolPanelHandler`, который создаётся с помощью `axipy.ActiveToolPanel.make_custom()`.

Список 3: Создание обработчика для взаимодействия с панелью активного инструмента.

```
service = ActiveToolPanel()
# Любой пользовательский графический элемент
widget = QWidget()

# Создаём обработчик для панели активного инструмента через который
# будем управлять панелью.
tool_panel = service.make_acceptable(
    title="Мой инструмент",
    observer_id=DefaultKeys.SelectionEditable,
    widget=widget)

# Подписываемся на сигнал отправляемый после нажатия на кнопку "Применить" в панели
tool_panel.accepted.connect(lambda: print("Применяем изменения"))
```

Чтобы показать панель активного инструмента с переданным ранее графическим элементом нужно вызвать `axipy.AxipyActiveToolPanelHandlerBase.activate()`, а для закрытия `axipy.AxipyActiveToolPanelHandlerBase.deactivate()`.

Список 4: Включение/Выключение панели активного инструмента.

```
class PyTool(MapTool):

    def mousePressEvent(self, event: QMouseEvent) -> Optional[bool]:
        if event.button() == Qt.LeftButton:
            self.tool_panel.activate()
            return self.PassEvent

    def keyPressEvent(self, event: QKeyEvent) -> Optional[bool]:
        if event.key() == Qt.Key_Escape:
            self.tool_panel.deactivate()
            return self.BlockEvent
        return super().keyPressEvent(event)
```

См.также:

Создание и добавление кнопок [Создание кнопок](#)

В коде выше в методе `mousePressEvent()` при нажатии левой кнопкой мыши мы показываем панель с активным инструментом, а в методе `keyPressEvent()` при нажатии на клавишу Esc панель закрываем.

Работа с длительными операциями

14.1 Задачи

Скрипты на python выполняются в основном потоке приложения или по другому в потоке интерфейса. Если операция будет выполняться слишком долго, то интерфейс «зависнет» и будет невозможно со стороны пользователя понять это ошибка или программа все-таки продолжает выполнять свою работу. Чтобы этого избежать длительные вычисления следует выносить в фоновые потоки. В потоке интерфейса во время выполнения фоновой задачи есть возможность показывать прогресс операции.

Чтобы превратить пользовательскую функцию в задачу, нужно использовать класс `axipy.Task`.

Список 1: Пример использования.

```
import time
import axipy

def user_heavy_function(t: axipy.Task, arg1: int, arg2: str):
    print(f"Переданные аргументы: {arg1}, {arg2}.")
    time.sleep(2)
    return 1

task = axipy.Task(user_heavy_function)
result = task.run_and_get("Hello", "world")
print(result)

"""
>>> Переданные аргументы: Hello, world.
>>> 1
"""
```

При использовании метода `axipy.Task.start()`, для получения результата, нужно подписаться на сигнал `axipy.Task.finished`.

Список 2: Пример использования.

```
import time
```

(continues on next page)

(продолжение с предыдущей страницы)

```
import axipy

def user_heavy_function(t: axipy.Task, arg1: int, arg2: str):
    print(f"Переданные аргументы: {arg1}, {arg2}.")
    time.sleep(2)
    return 1

task = axipy.Task(user_heavy_function)

def finished(t: axipy.Task):
    print(t.status)
    print(t.result)

task.finished.connect(finished)
task.start("Hello", "world")

"""
>>> Переданные аргументы: Hello, world.
>>> Status.SUCCESS
>>> 1
"""
```

Чтобы сделать поддержку типизированных аргументов и возвращаемого значения для задачи, рекомендуется наследование от базового класса `axipy.Task`.

Список 3: Пример использования.

```
import axipy

class SummTask(axipy.Task):

    def __init__(self, *args, **kwargs) -> None:
        super().__init__(self.run, *args, **kwargs)

    def run(self, arg1: float, arg2: float) -> float:
        self.value += 1
        return arg1 + arg2

    def run_and_get(self, arg1: float, arg2: float) -> float:
        return super().run_and_get(arg1, arg2)

result = SummTask().run_and_get(1.0, 2.3)
print(result)

"""
>>> 3.3
"""
```


14.2 Представление прогресса операции

Класс `axipy.DialogTask` дополняет класс `axipy.Task` наличием диалога для отображения прогресса операции. Для обмена информацией между выполняемой задачей и элементом, отображающим прогресс, используется ссылка на экземпляр задачи (`axipy.Task`), передаваемая первым аргументом в пользовательскую функцию. Типичный вариант использования выглядит следующим образом:

```
import axipy
import time

def user_heavy_function(dt: axipy.DialogTask, loop_range: int):
    dt.range = dt.Range(0, loop_range)
    dt.message = 'Прогресс выполнения'
    for i in range(loop_range):
        time.sleep(0.5)
        dt.value += 1
        if dt.is_canceled:
            return False
    return True

dialog_task = axipy.DialogTask(user_heavy_function)
dialog_task.title = 'Заголовок'
result = dialog_task.run_and_get(10)
print(result)
"""
>>> Result: True
"""
```

Вместо `axipy.Task.is_canceled` можно использовать `axipy.Task.raise_if_canceled` если нужно выйти из нескольких вложенных вызовов функций или циклов.

14.3 Создание пользовательского виджета для отображения прогресса

```
import axipy
import time
import PySide2.QtWidgets
import PySide2.QtCore

task_max_range = 20

def heavy_function(t: axipy.Task):
    for i in range(task_max_range):
        time.sleep(0.5)
        t.value += 1
        t.raise_if_canceled()
    return 1

class CustomDialog(PySide2.QtWidgets.QDialog):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
```

(continues on next page)

```

vbox = PySide2.QtWidgets.QVBoxLayout()
self.pbar = PySide2.QtWidgets.QProgressBar()
self.pbar.setRange(0, task_max_range)
vbox.addWidget(self.pbar)

bbox = PySide2.QtWidgets.QDialogButtonBox()
bbox.setStandardButtons(
    PySide2.QtWidgets.QDialogButtonBox.Cancel
)
bbox.rejected.connect(self.bbox_reject)
vbox.addWidget(bbox)

self.setLayout(vbox)

@PySide2.QtCore.Slot()
def bbox_reject(self) -> None:

    def ask() -> PySide2.QtWidgets.QMessageBox.StandardButton:
        return PySide2.QtWidgets.QMessageBox.question(
            axipy.view_manager.global_parent,
            "Отмена",
            "Отменить задачу?",
        )

    answer = ask()
    if answer == PySide2.QtWidgets.QMessageBox.Yes:
        self.reject()

@PySide2.QtCore.Slot()
def add_progress(self) -> None:
    self.pbar.setValue(self.pbar.value() + 1)

dialog = CustomDialog(axipy.view_manager.global_parent)
task = axipy.Task(heavy_function)
task.value_changed.connect(dialog.add_progress)
task.finished.connect(dialog.close)
dialog.rejected.connect(task.cancel)
task.start()
if task.status not in (
    task.Status.SUCCESS,
    task.Status.CANCELED,
    task.Status.ERROR,
):
    dialog.exec_()

```

14.4 Выполнение задач и многопоточность

Важно понимать, что задачи будут выполняться не в потоке интерфейса, поэтому внутри этих задач нельзя отображать никакие графические элементы (`PySide2.QtWidgets.QWidget`). Так же нужно внимательно следить за тем какие ресурсы могут использоваться несколькими потоками и при необходимости использовать различные механизмы синхронизации (мьютексы, локи и т.д.). Общее правило при работе с несколькими потоками следующее: старайтесь чтобы каждая задача содержала в себе все необходимые данные для выполнения. Синхронизацию, если она необходима, следует использовать только в момент получения результата. Это упростит код и сведет к минимум количество ошибок.

Переданные на выполнения задачи ставятся в общую очередь, которая распределяется между физическими ядрами процессора в порядке добавления. Поэтому нет гарантии, что переданная задача выполнится мгновенно, а так же то что группа задач будет выполняться именно в порядке добавления. Если задача предполагает долгое ожидание без интенсивных нагрузок на процессор, то лучше воспользоваться стандартными python потоками. Типичные примеры таких задач это загрузка ресурсов с диска или скачивание файлов по сети.

Разработка Плагинов (Модулей)

Плагин - это компонент, добавляющий определенный функционал в программу. Это позволяет программе быть расширяемой.

Данный раздел описывает требования и рекомендации по созданию таких компонентов для ГИС Аксиома.

Чтобы создать плагин, руководствуйтесь следующим:

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

Совет: Изучайте исходный код плагинов встроенных в Аксиому.

15.1 Структура плагина

Плагин для ГИС Аксиома - это специально оформленный модуль Python с дополнительными файлами.

Рассмотрим структуру минимального плагина с обязательными параметрами и более расширенного:

Минимальный:

```
ru_mycompany_minimal_module # папка плагина
├─ __init__.py # точка входа
└─ manifest.ini # информация о плагине
```

Расширенный:

```
ru_mycompany_extended_module
├── documentation
│   └── index.html
├── business_logic.py
├── i18n
│   ├── translation_en.qm
│   └── translation_en.ts
├── __init__.py
├── manifest.ini
└── ui
    ├── form.ui
    ├── image.png
    └── logo.png
```

15.1.1 Идентификатор плагина

`ru_mycompany_minimal_module` - папка с плагином, она же - уникальный идентификатор плагина. Так же, как и при создании обычных модулей Python, избегайте конфликтов имен. Делайте имя плагина 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 плагина `mymodule` рекомендуемым идентификатором будет: - для веб-сайта `axioma-gis.ru` - `ru_axioma_gis_mymodule` - для почты `andrey@yandex.ru` - `ru_yandex_at_andrey_mymodule`

15.1.2 Точка входа

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

См.также:

Точка входа должна содержать пользовательский класс, наследуемый от класса [Plugin](#).

15.1.3 Информация о плагине

`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:

```
name=Пример плагина
description=Короткий текст с описанием плагина.
```

См.также:

Для более подробного описания формата ini, поддерживаемого ГИС Аксиома, смотрите документацию [configparser](#).

Таблица 1: Таблица значений

Параметр	Обязательно	Описание
name	True	Короткая строка с именем модуля.
description	True	Короткий текст с описанием.
version	False	Строка с версией модуля.
homepage	False	Ссылка на домашнюю страницу модуля.
target	False	Версия Аксиомы, с которой работает модуль; цифры, разделенные точкой.
platforms	False	Список поддерживаемых платформ; через запятую из возможных Windows Linux и Darwin.
icon	False	Имя файла или относительный путь (относительно корневой папки модуля) в формате PNG.

Также могут содержаться другие необязательные параметры:

```
; может содержать комментарии
name=Пример модуля
description=Короткий текст с описанием модуля.
    Может быть многострочным.
; конец обязательных параметров

; необязательные параметры
version=1.0
homepage=https://dev.axioma-gis.ru
platforms=Windows,Darwin
target=3.1.0
author=Developer Name
email=dev@axioma-gis.ru
repository=https://github.com/developer/mymodule
license=New BSD
```

(continues on next page)

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

15.1.4 Документация

Документация может быть написана в HTML файлах. ГИС Аксиома откроет документацию в системном веб-браузере. Аксиома ищет документацию в папке с плагином `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`.

15.1.5 Переводы

Можно предусмотреть загрузку плагина на разных языках.

Название и описание самого плагина 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.

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 плагина. Они будут загружены вместе с плагином.

15.2 Класс Plugin

Модули пишутся в объектном стиле. Для этого, файл `__init__.py` должен содержать класс, наследуемый от класса `axipy.Plugin`. Тогда ГИС Аксиома при загрузке модуля создаст его экземпляр, а при выгрузке - удалит его.

Вспомогательный класс `axipy.Plugin` содержит свойства и методы, которые нужны для написания плагина. Например: загрузка/сохранение настроек `axipy.Plugin.settings`; получение пути к папке с модулем `axipy.Plugin.plugin_dir`; перевод строк `axipy.Plugin.tr()`; добавление кнопок в интерфейс `axipy.Plugin.create_action` и другие.

Пример модуля `__init__.py`

```
"""
Пример добавления кнопки и подключение действия по нажатию на нее (показ сообщения).
При выгрузке кнопка удаляется из интерфейса.
"""
from axipy import Position, Plugin, Notifications

class ExamplePluginMinimal(Plugin):
    def __init__(self) -> None:
        self._title = self.tr("Минимальный плагин")

        self._action = self.create_action(
            "Пример действия",
            icon="://icons/share/32px/run3.png",
            on_click=self.show_message,
        )
        position = Position("Примеры модулей", "Минимальный")
        position.add(self._action)
        self._action.action.setToolTip("Всплывающая подсказка")

    def unload(self) -> None:
        self._action.remove()

    def show_message(self) -> None:
        Notifications.push(self._title, "Пример выполнения действия по нажатию кнопки
→")
```

- При загрузке Аксиома создаст экземпляр модуля и вызовет конструктор класса `axipy.Plugin.__init__()`.
- После создания главного окна, Аксиома вызовет метод `axipy.Plugin.load()`.
- `axipy.Plugin.unload()` - вызывается, когда модуль выгружается.

15.3 Архив

Физически плагин представлен в виде папки с уникальным именем, внутри которой расположены файлы и папки с бизнес-логикой, конфигурациями, документацией, зависимостями, графическими формами и прочим. Для гарантии целостности и удобства распространения готовые плагины помещаются в архив.

Архив использует формат [ZIP](#) и имеет следующую структуру:

```
my_plugin_archive_v1.ахр
├─ ru_axioma_gis_axipy_example_plugin_from_package
│   └─ __init__.py
│   └─ manifest.ini
```

Таким образом архив просто содержит папку с плагином. Имя архива может быть любым и должно заканчиваться на .ахр, в то время как имя папки с плагином должно быть уникальным.

Для создания архива достаточно запаковать плагин в ZIP любым поддерживаемым архиватором и указать расширение выходного файла как .ахр вместо стандартного .zip.

Архив с плагином распаковывается в пользовательскую директорию при установке. Архив может быть установлен пользователем через интерфейс программы ГИС Аксиома в диалоге «Модули». Плагины, установленные пользователем, могут быть удалены из того же диалога.

Физически плагины устанавливаются в installed_modules в пользовательскую папку. Расположение пользовательской папки зависит от операционной системы:

Операционная система	Пользовательская папка Аксиомы
Windows	%APPDATA%\ESTI\Axioma.GIS
Linux	\$HOME/.local/share/ESTI/Axioma.GIS/
macOS	\$HOME/Library/Application Support/ESTI/Axioma.GIS/

15.4 Зависимости

15.4.1 Зависимости плагинов. Установка зависимостей.

Плагины Аксиомы могут использовать сторонние библиотеки Python. Такой плагин во время установки обратится к каталогу пакетов [PyPI](#) и установит необходимые зависимости.

В коде плагина пакет импортируется обычным способом:

Список 1: Файл __init__.py

```
import numpy
...
```

Зависимости перечисляются в специальном файле requirements.txt. Так плагин с зависимостями может иметь следующую структуру:

```
ru_axioma_gis_axipy_example_plugin_from_package
├── __init__.py
├── manifest.ini
└── requirements.txt
```

Файл является простым текстовым файлом в кодировке UTF-8, в котором построчно перечислены необходимые пакеты. Например:

Список 2: Файл requirements.txt

```
numpy
requests
idna
```

Примечание: В настоящий момент нет возможности указать версию пакета.

Также, зависимости можно установить напрямую, например, для использования в интерактивной консоли python.

Пример команды установки зависимостей из файла «requirements.txt»:

Список 3: Команда для установки зависимостей из файла «requirements.txt», где <AXIOMA> - папка установленной Аксиомы.

```
# Linux
"<AXIOMA>/sbin/run-python" -m pip install -r ./requirements.txt
# Например:
"/opt/axioma-gis6/sbin/run-python" -m pip install -r ./requirements.txt

# Windows
"<AXIOMA>\bin\python\python.exe" -m pip install -r requirements.txt
# Например:
"C:\Program Files\Axioma v6\bin\python\python.exe" -m pip install -r requirements.txt
```

Пример команды установки отдельного пакета по имени:

Список 4: Команда для установки отдельного пакета по имени, где <AXIOMA> - папка установленной Аксиомы.

```
"C:/Program Files/Axioma v4/bin/python/python.exe" -m pip install qt5_applications

# Linux
"<AXIOMA>/sbin/run-python" -m pip install qt5_applications
# Например:
"/opt/axioma-gis6/sbin/run-python" -m pip install qt5_applications

# Windows
"<AXIOMA>\bin\python\python.exe" -m pip install qt5_applications
# Например:
"C:\Program Files\Axioma v6\bin\python\python.exe" -m pip install qt5_applications
```

где qt5_applications наименование пакета, который необходимо установить.

15.4.2 Расположение каталога установки зависимостей site-packages

Расположение каталога установки зависимостей, где <VERSION> - это номер версии Аксиомы, например «6».

Для Windows:

```
"%APPDATA%/ESTI/Axioma.GIS/v<VERSION>/installed_modules/dependencies/  
site-packages"
```

Примерный результат:

```
C:/Users/user/AppData/Roaming/ESTI/Axioma.GIS/v6/installed_modules/  
dependencies/site-packages
```

Для Linux:

```
"${HOME}/.local/share/ESTI/Axioma.GIS/v<VERSION>/installed_modules/  
dependencies/site-packages"
```

Примерный результат:

```
/home/user/.local/share/ESTI/Axioma.GIS/v6/installed_modules/dependencies/  
site-packages
```

Для macOS:

```
"${HOME}/Library/Application  
Support/ESTI/Axioma.GIS/v6/installed_modules/dependencies/site-packages"
```

Примерный результат:

```
/home/user/Library/Application  
Support/ESTI/Axioma.GIS/v6/installed_modules/dependencies/site-packages
```

Также, определить каталог, куда будут устанавливаться зависимые python пакеты можно следующей командой:

Список 5: Команда для получения каталога зависимых пакетов, где <AXIOMA> - папка установленной Аксиомы.

```
# Linux  
"<AXIOMA>/sbin/run-python" -c "import axipy; axipy.init_axioma(); print(axipy.get_  
↳dependencies_folder())"  
# Например:  
"/opt/axioma-gis6/sbin/run-python" -c "import axipy; axipy.init_axioma(); print(axipy.  
↳get_dependencies_folder())"  
  
# Windows  
"<AXIOMA>\bin\python\python.exe" -c "import axipy; axipy.init_axioma(); print(axipy.  
↳get_dependencies_folder())"  
# Например:  
"C:\Program Files\Axioma v6\bin\python\python.exe" -c "import axipy; axipy.init_  
↳axioma(); print(axipy.get_dependencies_folder())"
```

15.4.3 Без интернета. Ручная установка пакетов.

Может возникнуть ситуация, когда устанавливаемый плагин имеет внешние зависимости, а доступа к каталогу пакетов PyPI нет. Плагин не сможет успешно установиться.

В этом случае можно скачать все необходимые зависимые пакеты на компьютере, который имеет доступ к интернету и конкретно к каталогу пакетов PyPI, а затем перенести их на целевой компьютер.

Для этого рекомендуется:

1. Установить ГИС Аксиома на компьютер с доступом к сети Интернет (той же версии и платформы).
2. Извлечь из архива с плагином .ахр файл зависимостей requirements.txt.
3. Используя командную строку и интерпретатор Python внутри установленной Аксиомы, выполнить

Список 6: Загрузка необходимых пакетов в папку, где
<AXIOMA> - папка установленной Аксиомы.

```
# Linux
"<AXIOMA>/sbin/run-python" -m pip download -r ./requirements.txt --dest ./module_deps/
# Например:
"/opt/axioma-gis6/sbin/run-python" -m pip download -r ./requirements.txt --dest ./
↳ module_deps/

# Windows
"<AXIOMA>\bin\python\python.exe" -m pip download -r requirements.txt --dest module_
↳ deps
# Например:
"C:\Program Files\Axioma v6\bin\python\python.exe" -m pip download -r requirements.
↳ txt --dest module_deps
```

, где module_deps - папка, в которую будут загружены зависимые пакеты.

4. Перенести зависимые пакеты на компьютер без интернета.
5. Аналогично, используя командную строку и интерпретатор Python внутри установленной Аксиомы, выполнить

Список 7: Установка необходимых пакетов из папки,
где <AXIOMA> - папка установленной Аксиомы.

```
# Linux
"<AXIOMA>/sbin/run-python" -m pip install -r ./requirements.txt --no-index --find-
↳ links ./module_deps/
# Например:
"/opt/axioma-gis6/sbin/run-python" -m pip install -r ./requirements.txt --no-index --
↳ find-links ./module_deps/

# Windows
"<AXIOMA>\bin\python\python.exe" -m pip install -r requirements.txt --no-index --find-
↳ links module_deps
# Например:
"C:\Program Files\Axioma v6\bin\python\python.exe" -m pip install -r requirements.txt
↳ --no-index --find-links module_deps
```

6. Установить архив с плагином *.ахр.

15.5 Рекомендации по написанию плагинов

Импорты

При импорте рекомендуется использовать конструкцию, наподобие следующей, где перечислены все необходимые имена:

```
from axipy import (Position, ObserverManager, Notifications, Plugin, Separator,   
↳ ActionButton, ToolButton, Button,   
    Geometry, Style, CoordSystem)
```

Предупреждение: В соответствии с **PEP 8#Imports**, не рекомендуется делать импорт всей библиотеки.

```
from axipy import *
```

При использовании большого количества библиотек, чтобы избежать конфликта имен, рекомендуется использовать конструкцию:

```
import axipy as axp  
  
feature = axp.Feature({'attr_name': 'value'}, geometry=axp.Point(10, 10))
```

Создание приложения

В данном разделе рассмотрим создание простейшего приложения, которое можно запускать из оболочки `python`. При этом пользовательский интерфейс будет строиться исходя из предпочтений пользователя. В рамках примера создадим приложение, которое будет открывать файл и показывать его в окне карты. Также вставим стандартное окно управления слоями карты `axipy.LayerControlWidget`.

Примечание: Для использования библиотеки `axipy` без интерфейса Аксиомы требуется платная лицензия.

Первым шагом проинициализируем ядро (подробнее см. [Initialization, Finalization, and Threads](#)) и определим главное окно нашего будущего приложения. Код будет выглядеть следующим образом:

```
from PySide2.QtWidgets import QMainWindow
from axipy import init_axioma

class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Пример окна приложения")

# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()
```

После того как мы создали пустое окно добавим панель инструментов со стандартными действиями (полный список действий можно посмотреть `axipy.ActionManager.items()`). Код будет выглядеть следующим образом:

```
from PySide2.QtWidgets import QMainWindow
from axipy import init_axioma, ActionManager

class MainWindow(QMainWindow):

    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Пример окна приложения")
        self._add_buttons()

    def _add_buttons(self) -> None:
        def _add_action(name) -> None:
            # Добавляем системное действие
            act = ActionManager.get(name)
            if act:
                self.toolbar.addAction(act)

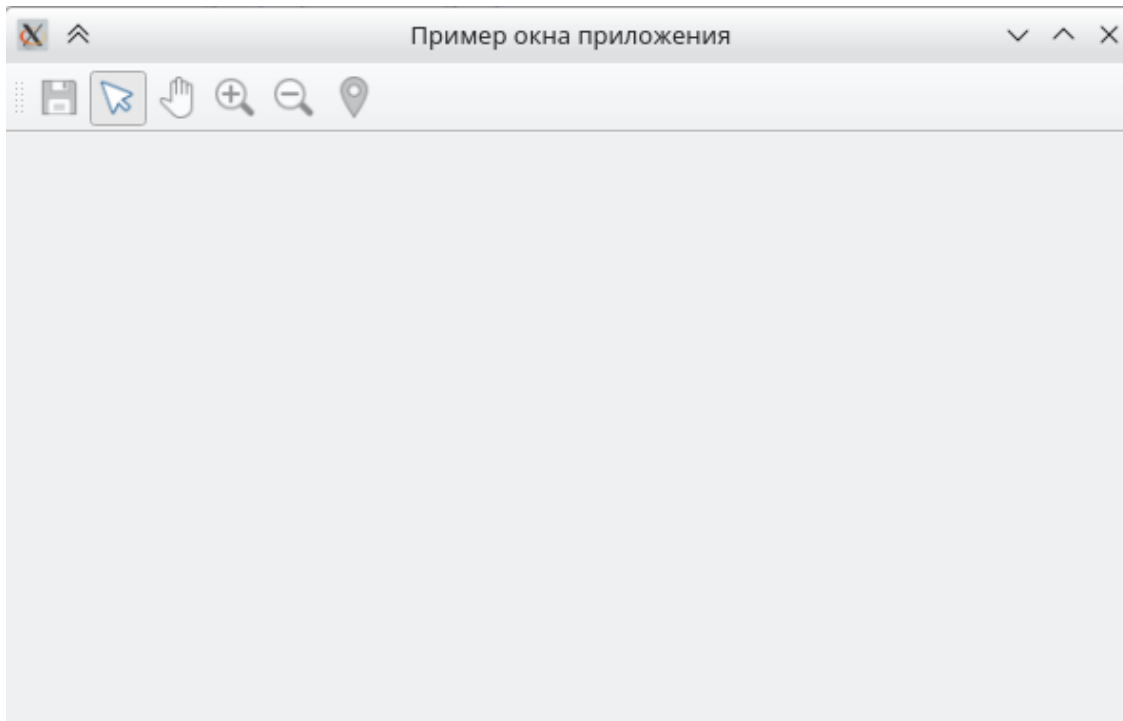
        # Создаем панель инструментов
        self.toolbar = self.addToolBar("Панель")
        # Добавляем стандартные кнопки
        _add_action("SaveTables")
        _add_action("Select")
        _add_action("Pan")
        _add_action("ZoomIn")
        _add_action("ZoomOut")
        _add_action("PointDraw")

# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()
```

После запуска окно будет выглядеть примерно так:



Далее, нам необходимо добавить пользовательское действие и, как реакция на него, будет открытие и отображение файла в окне карты. Одновременно с этим создается окно управления слоями.

Результирующий код будет выглядеть следующим образом:

```
from PySide2.QtCore import Qt
from PySide2.QtWidgets import QMainWindow, QFileDialog, QDockWidget, QAction
from axipy import (
    init_axioma, provider_manager, Layer, view_manager, Map, LayerControlWidget,
    ↪ActionManager
)

class MainWindow(QMainWindow):

    def __init__(self) -> None:
        super().__init__()
        self.setWindowTitle("Пример окна приложения")
        self._add_buttons()

    # Открытие файла
    def __open_file(self) -> None:
        file_name, _ = QFileDialog.getOpenFileName(self, "Открыть", filter="MapInfo_
        ↪TAB (*.tab)")
        if file_name:
            table = provider_manager.openfile(file_name)
            # создаем слой
            layer = Layer.create(table)
            # если карты нет, то создаем ее
            if not view_manager.active:
                map_ = Map([layer])
                mapview = view_manager.create_mapview(map_)
```

(continues on next page)

```

        self._create_layer_control_dock()
        view_manager.activate(mapview)
        self.setCentralWidget(mapview.widget)
        # если есть, добавляем слой в окно карты
        else:
            view_manager.active.map.layers.append(layer)

# Добавление по левой стороне окно управления слоями карты
def _create_layer_control_dock(self) -> None:
    self._layer_control_w = LayerControlWidget()
    dock_layer_control = QDockWidget(self._layer_control_w.widget.windowTitle(),
↪self)
    dock_layer_control.setWidget(self._layer_control_w.widget)
    self.addDockWidget(Qt.LeftDockWidgetArea, dock_layer_control)

# Добавление кнопок на панель инструментов
def _add_buttons(self) -> None:
    # Создаем панель инструментов
    self.toolbar = self.addToolBar("Панель")

    # Пользовательское действие открытия файла
    self.action_open = QAction(ActionManager.icon_by_name("open"), "Открыть файл")
    self.action_open.triggered.connect(self.__open_file)
    self.toolbar.addAction(self.action_open)

    def _add_action(name) -> None:
        # Добавляем системное действие
        act = ActionManager.get(name)
        if act:
            self.toolbar.addAction(act)

    _add_action("SaveTables")
    _add_action("Select")
    _add_action("Pan")
    _add_action("ZoomIn")
    _add_action("ZoomOut")
    _add_action("PointDraw")

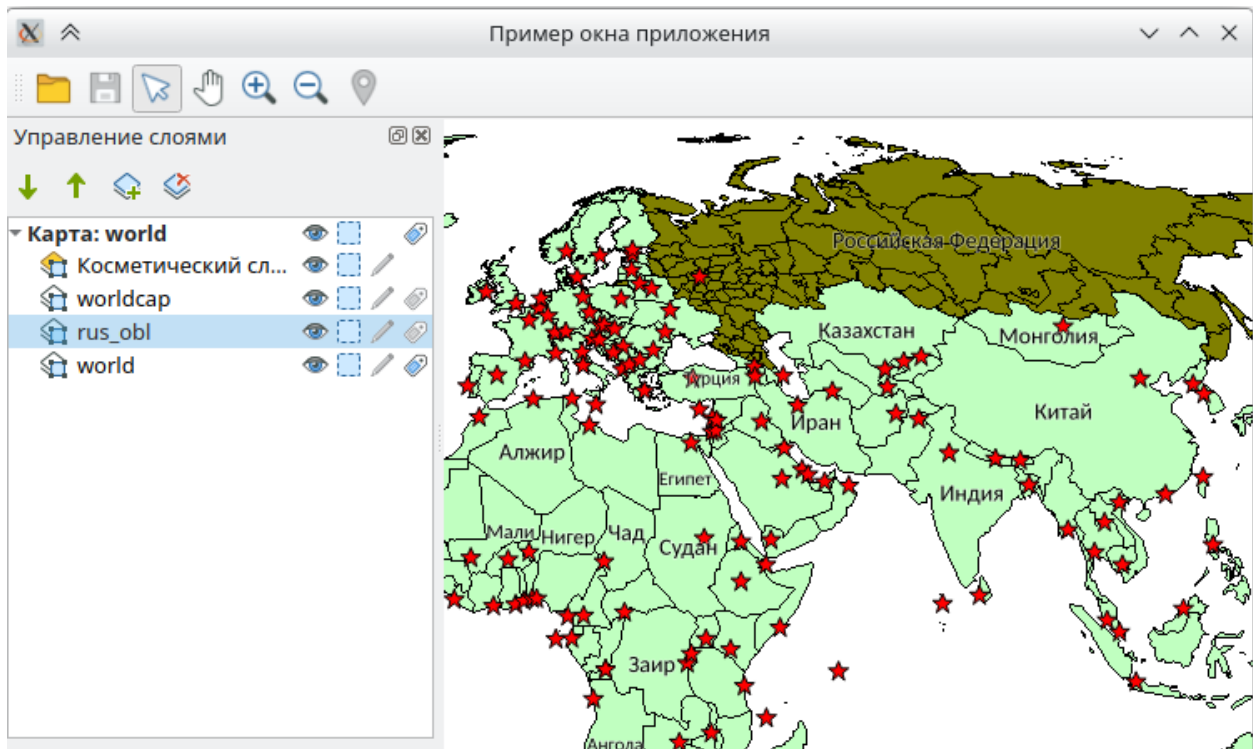
# Инициализация ядра
app = init_axioma()

# Создание и отображение главного окна
main_window = MainWindow()
main_window.resize(1200, 800)
main_window.show()

# Запуск основного цикла программы
app.exec_()

```

Результат работы:



16.1 Пример приложения из готовых компонентов

Следующим этапом попробуем усложнить пример и сделать приложение, которое содержит помимо окна карты и окна с деревом слоев и другие. Перечислим их:

- LayerControlWidget - Виджет управления слоями карты.
- ViewManagerWidget - Виджет списка окон.
- DataManagerWidget - Виджет управления списком открытых данных.
- PythonConsoleWidget - Виджет ввода команд python.
- NotificationWidget - Виджет уведомлений.

```
from PySide2.QtCore import Qt, Slot
from PySide2.QtWidgets import QMainWindow, QDockWidget, QStackedWidget

import os
from axipy import (Notifications, LayerControlWidget, DataManagerWidget,
↳ NotificationWidget, ViewManagerWidget,
↳ PythonConsoleWidget, view_manager, init_axioma, provider_manager,
↳ Layer, Map, View, ActionManager)

class MainWindow(QMainWindow):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("Пример окна приложения")
```

(continues on next page)

```

self._stacked_widget = QStackedWidget()
view_manager.active_changed.connect(self.set_stacked_widget)

self.setCentralWidget(self._stacked_widget)
self.__initButtons()

def _add_action(self, id):
    act = ActionManager[id]
    if act:
        self.toolbar.addAction(act)
        self.menu.addAction(act)

def __initButtons(self):
    self.toolbar = self.addToolBar('Панель')
    self.menu = self.menuBar().addMenu('Меню')
    self._add_action('Select')
    self._add_action('SelectInRect')
    self._add_action('Pan')
    self._add_action('ZoomIn')
    self._add_action('ZoomOut')
    self.toolbar.addSeparator()
    self._add_action('PointDraw')
    self._add_action('RectangleDraw')
    self._add_action('SymbolStyle')
    ActionManager.activate("Pan")

def _create_maps_dock(self):
    self._layer_control_w = LayerControlWidget()
    self._layer_control_w.mapview_activated.connect(self.set_center_widget)
    dock_layer_control = QDockWidget(self._layer_control_w.widget.windowTitle(),
↪main_window)
    dock_layer_control.setWidget(self._layer_control_w.widget)
    self.addDockWidget(Qt.LeftDockWidgetArea, dock_layer_control)

def _create_data_manager_dock(self):
    data_manager_w = DataManagerWidget()
    dock_catalog = QDockWidget(data_manager_w.widget.windowTitle())
    dock_catalog.setWidget(data_manager_w.widget)

    def notify_selection():
        """
        Отслеживание выборки в списке открытых данных.
        """
        objects = data_manager_w.objects
        if len(objects) > 0:
            names = ', '.join(obj.name for obj in objects)
            Notifications.push("Выбрано", names, Notifications.Warning)

    data_manager_w.selection_changed.connect(notify_selection)
    self.addDockWidget(Qt.LeftDockWidgetArea, dock_catalog)

def _create_notification_dock(self):
    notification_w = NotificationWidget()
    notification_w.widget.setFixedHeight(150)
    dock_notification = QDockWidget(notification_w.widget.windowTitle())

```

(continues on next page)

(продолжение с предыдущей страницы)

```

dock_notification.setWidget(notification_w.widget)
self.addDockWidget(Qt.BottomDockWidgetArea, dock_notification)

def _create_python_dock(self):
    notification_w = PythonConsoleWidget()
    notification_w.widget.setFixedHeight(150)
    dock_notification = QDockWidget(notification_w.widget.windowTitle())
    dock_notification.setWidget(notification_w.widget)
    self.addDockWidget(Qt.BottomDockWidgetArea, dock_notification)

def _create_view_manager_dock(self):
    self._view_manager_w = ViewManagerWidget()
    self._view_manager_w.view_changed.connect(self.set_center_widget)
    dock_widget = QDockWidget(self._view_manager_w.widget.windowTitle(), main_
↪window)
    dock_widget.setWidget(self._view_manager_w.widget)
    self.addDockWidget(Qt.LeftDockWidgetArea, dock_widget)

def init_gui(self):
    # Окно управления слоями
    self._create_maps_dock()
    # Открытые источники
    self._create_data_manager_dock()
    # Окно уведомлений
    self._create_notification_dock()
    # Окно консоли python
    self._create_python_dock()
    # Список окон
    self._create_view_manager_dock()

@Slot(View)
def set_center_widget(self, inp_view: View):
    """
    Устанавливает активное окно.
    """
    view_manager.activate(inp_view)

@Slot()
def set_stacked_widget(self):
    w = view_manager.active.widget
    i = self._stacked_widget.indexOf(w)
    if i == -1:
        self._stacked_widget.insertWidget(0, w)
    else:
        self._stacked_widget.setCurrentIndex(i)

# Инициализация ядра
app = init_axioma()

# Создаем главное окно приложения
main_window = MainWindow()
main_window.init_gui()

# Открываем таблицы

```

(continues on next page)

(продолжение с предыдущей страницы)

```
def path_to_file(file_name: str) -> str:
    return os.path.join(os.path.dirname(__file__), "tab", file_name)

table_world = provider_manager.openfile(path_to_file("world.tab"))
table_world_cap = provider_manager.openfile(path_to_file("worldcap.tab"))
table_rf = provider_manager.openfile(path_to_file("SubjectRF.TAB"))

# Создаем слои
layer_world = Layer.create(table_world)
layer_world_cap = Layer.create(table_world_cap)
layer_rf = Layer.create(table_rf)

# Создаем окно карты
map_world_all = Map([layer_world_cap, layer_world])
mapview = view_manager.create_mapview(map_world_all)

# Еще одна карта
map_world = Map([layer_rf, layer_world])
view_manager.create_mapview(map_world)

# Таблицы просмотра
view_manager.create_tableview(table_world)
view_manager.create_tableview(table_rf)

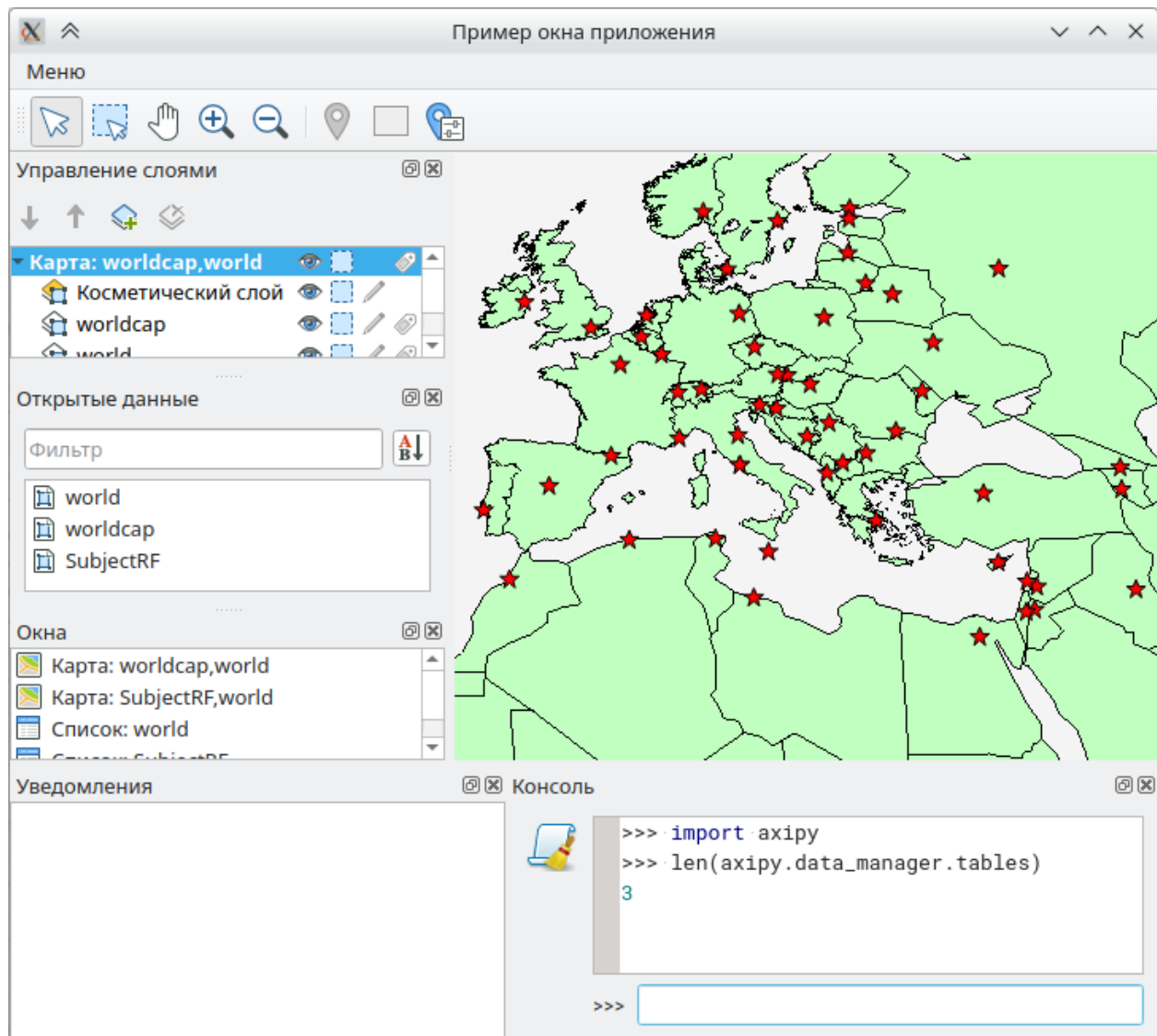
# Отчет
#view_manager.create_reportview()

# Первое окно как основное
main_window.set_center_widget(mapview)

# Левый верхний угол окна
x, y = 200, 10
# Ширина и высота окна
width, height = 1200, 800
main_window.resize(width, height)
main_window.move(x, y)

# Показываем главное окно
main_window.show()
app.exec_()
```

Результат работы:



axipy - Основной пакет библиотеки для взаимодействия с ГИС Аксиома.

Основной пакет API для взаимодействия с ГИС Аксиома.

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

17.1 Инициализация Аксиомы

class axipy.AxiomaInitLogLevel

Уровень логирования Аксиомы при запуске из python.

axipy.**init_axioma**(*, log_level: AxiomaInitLogLevel = AxiomaInitLogLevel.medium,
language: AxiomaLanguage = AxiomaLanguage.ru,
load_python_plugins: bool = False) → QApplication

Инициализирует ядро ГИС Аксиома.

Параметры

- **log_level** – Уровень логирования.
- **language** – Язык Аксиомы.
- **load_python_plugins** – Загружать плагины.

Результат

Приложение Qt5 с очередью событий (event-loop).

Пример:

```
from axipy import init_axioma, mainwindow, AxiomaInitLogLevel

app = init_axioma(log_level=AxiomaInitLogLevel.disabled)
mainwindow.show()
app.exec_() # запускает обработку очереди событий
```

17.2 Plugin - Плагин ГИС Аксиома

class ахіру.Plugin

Вспомогательный класс для создания плагинов.

Свойства:

<code>plugin_dir</code>	Возвращает путь к папке плагина.
<code>settings</code>	Настройки плагина.

Методы:

<code>create_action(title, on_click[, icon, ...])</code>	Создает кнопку с действием.
<code>create_tool(title, on_click[, icon, ...])</code>	Создает кнопку с инструментом.
<code>get_plugin_data_dir()</code>	Возвращает каталог, в котором находятся изменяемые данные плагина.
<code>load()</code>	Переопределите этот метод для задания логики загрузки плагина.
<code>tr(text)</code>	Ищет перевод строки в загруженных файлах перевода.
<code>unload()</code>	Переопределите этот метод для очистки ресурсов при выгрузке плагина.

create_action(title: str, on_click: Callable[[], Any], icon: Union[str, QIcon] = "", enable_on: Optional[Observer] = None, tooltip: Optional[str] = None, doc_file: Optional[str] = None) → [ActionButton](#)

Создает кнопку с действием.

Параметры

- **title** - Текст.
- **on_click** - Действие на нажатие.
- **icon** - Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** - Идентификатор наблюдателя для определения доступности кнопки.
- **tooltip** - Строка с дополнительной короткой информацией по данному действию.
- **doc_file** - Относительная ссылка на файл документации. Расположение рассматривается по отношению к каталогу documentation.

Результат

Кнопка с действием.

Примечание: То же, что и [ActionButton](#), но дополнительно делает идентификатор кнопки уникальным для данного плагина.

```
create_tool(title: str, on_click: Union[Callable[[], MapTool], MapTool], icon:
    Union[str, QIcon] = "", enable_on: Optional[Union[str, Observer]] =
    None, tooltip: Optional[str] = None, doc_file: Optional[str] = None) →
    ToolButton
```

Создает кнопку с инструментом.

Параметры

- **title** - Текст.
- **on_click** - Класс инструмента или же функция, возвращающая экземпляр инструмента.
- **icon** - Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** - Идентификатор наблюдателя для определения доступности кнопки.
- **tooltip** - Строка с дополнительной короткой информацией по данному действию.
- **doc_file** - Относительная ссылка на файл документации. Расположение рассматривается по отношению к каталогу documentation.

Результат

Кнопка с инструментом.

Примечание: То же, что и `ToolButton`, но дополнительно делает идентификатор кнопки уникальным для данного плагина.

Пример:

```
def create_tool() -> MapTool:
    return MapTool()

class ExampleTool(Plugin):
    def __init__(self) -> None:
        tool = self.create_tool(
            title="Пример инструмента",
            icon="://icons/share/32px/run3.png",
            on_click=create_tool
        )
```

get_plugin_data_dir() → `Path`

Возвращает каталог, в котором находятся изменяемые данные плагина.

load()

Переопределите этот метод для задания логики загрузки плагина.

property plugin_dir: `Path`

Возвращает путь к папке плагина.

property settings: `QSettings`

Настройки плагина.

Позволяет сохранять и загружать параметры.

См.также:

Подробнее в документации на класс `PySide2.QtCore.QSettings`.

`tr(text: str) → str`

Ищет перевод строки в загруженных файлах перевода.

Параметры

text – Строка для перевода.

Результат

Перевод строки, если строка найдена. Иначе — сама переданная строка.

Пример:

```
button_name = self.tr("My button")
```

`unload()`

Переопределите этот метод для очистки ресурсов при выгрузке плагина.

17.3 PluginManager - Менеджер плагинов

`class axipy.PluginManager`

Менеджер плагинов.

Примечание: Создание `axipy.PluginManager` не требуется, используйте объект `axipy.plugin_manager`.

Пример:

```
import axipy

# Отслеживание загрузки и выгрузки плагинов
axipy.plugin_manager.loaded.connect(lambda id_: print(f'Загружен плагин {id_}'))
axipy.plugin_manager.unloaded.connect(lambda id_: print(f'Выгружен плагин {id_}'))
plugin_name = 'my_module_id'
# Загрузка плагина
axipy.plugin_manager.load(plugin_name)
# Получение кортежа идентификаторов доступных плагинов
ids = axipy.plugin_manager.ids()
# Получение информации о плагине
pi = axipy.plugin_manager.info(plugin_name)
print(pi.name, pi.author, pi.path)
```

Методы:

<code>ids()</code>	Кортеж идентификаторов плагинов.
<code>info(plugin_id)</code>	Информация о плагине.
<code>is_loaded(plugin_id)</code>	Загружен ли плагин.
<code>load(plugin_id)</code>	Загрузка плагина.
<code>unload(plugin_id)</code>	Выгрузка плагина.

Сигналы:

<code>loaded</code>	Плагин успешно загружен.
<code>unloaded</code>	Плагин успешно выгружен.

`ids()` → `Tuple[str, ...]`

Кортеж идентификаторов плагинов.

`info(plugin_id: str)` → `PluginInfo`

Информация о плагине.

Параметры

`plugin_id` – Идентификатор плагина.

Исключение

`ValueError` – Если плагин не найден.

`is_loaded(plugin_id: str)` → `bool`

Загружен ли плагин.

Параметры

`plugin_id` – Идентификатор плагина.

Результат

True если плагин загружен. В противном случае False, также, если плагин отсутствует.

`load(plugin_id: str)`

Загрузка плагина. При ошибке выбрасывается исключение.

Параметры

`plugin_id` – Идентификатор плагина.

Исключение

`ValueError` – Если плагин не найден.

`property loaded: Signal`

Плагин успешно загружен.

Тип результата

`Signal[str]`, где `str` - идентификатор плагина.

`unload(plugin_id: str)`

Выгрузка плагина.

Параметры

`plugin_id` – Идентификатор плагина.

Исключение

`ValueError` – Если плагин не найден.

`property unloaded: Signal`

Плагин успешно выгружен.

Тип результата

`Signal[str]`, где `str` - идентификатор плагина.

17.4 PluginInfo - Информация о модуле

class axipy.PluginInfo

Информация о плагине.

См.также:

Менеджер плагинов [PluginManager](#)

Атрибуты:

author	Автор плагина.
description	Краткое описание.
documentation	Ссылка на документацию.
homepage	Домашняя страница.
name	Наименование.
path	Путь в файловой системе.
version	Версия.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

17.5 Настройки ГИС Аксиома

class axipy.CurrentSettings

Текущие настройки Аксиомы. Класс является статическим словарем (`collections.abc.MutableMapping`). Поддерживает обращение по индексу.

Примечание: Для получения настроек по умолчанию, используется класс `axipy.DefaultSettings`.

Список 1: Пример работы с настройками.

```
from axipy import DefaultSettings, CurrentSettings

# Получение настройки по умолчанию
print(DefaultSettings.ShowSplashScreen)
'''
>>> True
'''

# Назначение текущей настройки
CurrentSettings.ShowSplashScreen = False
# Получение текущей настройки
print(CurrentSettings.ShowSplashScreen)
'''
>>> False
'''

# Сброс настройки обратно к значению по умолчанию
```

(continues on next page)

(продолжение с предыдущей страницы)

```
CurrentSettings.ShowSplashScreen = DefaultSettings.ShowSplashScreen
print(CurrentSettings.ShowSplashScreen)
'''
>>> True
'''
```

Классовые методы:

<code>get(key[, default_value])</code>	Возвращает значение по ключу.
<code>items()</code>	Возвращает список кортежей ключ-значение, где ключи - это атрибуты класса, а значения - это значения настроек.
<code>keys()</code>	Возвращает список ключей, где ключи это атрибуты класса.
<code>reset()</code>	Сброс значений всех текущих настроек на значения по умолчанию.
<code>values()</code>	Возвращает список значений, где значения это значения настроек.

Атрибуты:

<code>BrushCatalog</code>	Каталог со стилями заливок.
<code>CreateTabAfterOpen</code>	Создавать ТАВ при открытии.
<code>DefaultPathCache</code>	Каталог с кэшированными данными.
<code>DistancePrecision</code>	Точность по умолчанию для расстояний и площадей.
<code>DrawCoordSysBounds</code>	Отображать границы мира.
<code>EditNodeColor</code>	Узлы при редактировании - цвет
<code>EditNodeSize</code>	Узлы при редактировании - размер
<code>EnableSmartTabs</code>	Умное переключение вкладок.
<code>Language</code>	Язык Аксиомы.
<code>LastNameFilter</code>	Последний использованный фильтр файлов.
<code>LastOpenPath</code>	Последний каталог откуда открывались данные.
<code>LastPathWorkspace</code>	Последний каталог к рабочему набору.
<code>LastSavePath</code>	Последний путь сохранения.
<code>LoadLastWorkspace</code>	Загружать при старте последнее рабочее пространство.
<code>MeshSizeLayout</code>	Размер ячейки.
<code>MeshSizeLegend</code>	Размер ячейки.
<code>NearlyGeometriesTopology</code>	Перемещать узлы соседних объектов при редактировании.
<code>NodesUpdateMode</code>	Объединять историю изменения узлов в режиме Форма.
<code>PenCatalog</code>	Каталог со стилями линий.
<code>PhysicalScreenSize</code>	Размер экрана в дюймах.
<code>PreserveScaleMap</code>	Сохранять масштаб при изменении размеров окна.

continues on next page

Таблица 1 – продолжение с предыдущей страницы

<code>RenameDataObjectFromTab</code>	Переименовывать открытый объект по имени TAB файла.
<code>RulerColorLine</code>	Линейка - цвет линии
<code>SaveAsToOriginalFileFolder</code>	Сохранять копию в каталог с исходным файлом.
<code>SelectByInformationTool</code>	Инструмент «Информация» выбирает объект.
<code>SensitiveMouse</code>	Чувствительность мыши в пикселях.
<code>ShowDegreeTypeNumeric</code>	Отображать градусы в формате Десятичное значение.
<code>ShowDrawingToolTip</code>	Показывать данные при рисовании.
<code>ShowMapScaleBar</code>	Показывать масштабную линейку.
<code>ShowMeshLayout</code>	Отображать сетку привязки.
<code>ShowMeshLegend</code>	Отображать сетку привязки.
<code>ShowScrollOnMapView</code>	Показывать полосы прокрутки.
<code>ShowSplashScreen</code>	Отображать экран загрузки.
<code>SilentCloseWidget</code>	Подтверждать закрытие несохраненных данных.
<code>SnapSensitiveRadius</code>	Привязка узлов - размер
<code>SnapToMeshLayout</code>	Привязывать элементы отчета к сетке.
<code>SnapToMeshLegend</code>	Привязывать к сетке.
<code>SymbolCatalog</code>	Каталог со стилями точек.
<code>UseAntialiasing</code>	Использовать сглаживание при отрисовке.
<code>UseLastSelectedFilter</code>	Запоминать последний фильтр в диалоге открытия файлов.
<code>UseNativeFileDialog</code>	Пользоваться системными файловыми диалогами.
<code>UserDataFolder</code>	Папка с пользовательскими данными Аксиомы.
<code>UserDataPaths</code>	Список пользовательских каталогов с названиями.
<code>UserPluginsDataFolder</code>	Папка с данными установленных плагинов Аксиомы.
<code>UserPluginsDependenciesFolder</code>	Папка с зависимостями установленных плагинов Аксиомы.
<code>UserPluginsFolder</code>	Корневая папка установленных плагинов Аксиомы.
<code>UserPluginsInstallationFolder</code>	Папка с установленными плагинами Аксиомы.
<code>UserPluginsSettingsFolder</code>	Папка с настройками установленных плагинов Аксиомы.

BrushCatalog: Path

Каталог со стилями заливок.

CreateTabAfterOpen: bool

Создавать TAB при открытии.

DefaultPathCache: Path

Каталог с кэшированными данными.

DistancePrecision: int

Точность по умолчанию для расстояний и площадей.

DrawCoordSysBounds: bool

Отображать границы мира.

EditNodeColor: QColor

Узлы при редактировании - цвет

EditNodeSize: int

Узлы при редактировании - размер

EnableSmartTabs: bool

Умное переключение вкладок.

Language: AxiomaLanguage

Язык Аксиомы.

LastNameFilter: str

Последний использованный фильтр файлов.

LastOpenPath: Path

Последний каталог откуда открывались данные.

LastPathWorkspace: Path

Последний каталог к рабочему набору.

LastSavePath: Path

Последний путь сохранения.

LoadLastWorkspace: bool

Загружать при старте последнее рабочее пространство.

MeshSizeLayout: float

Размер ячейки.

MeshSizeLegend: float

Размер ячейки.

NearlyGeometriesTopology: bool

Перемещать узлы соседних объектов при редактировании.

NodesUpdateMode: bool

Объединять историю изменения узлов в режиме Форма.

PenCatalog: Path

Каталог со стилями линий.

PhysicalScreenSize: float

Размер экрана в дюймах.

PreserveScaleMap: bool

Сохранять масштаб при изменении размеров окна.

RenameDataObjectFromTab: bool

Переименовывать открытый объект по имени ТАВ файла.

RulerColorLine: QColor

Линейка - цвет линии

SaveAsToOriginalFileFolder: bool

Сохранять копию в каталог с исходным файлом.

SelectByInformationTool: bool

Инструмент «Информация» выбирает объект.

SensitiveMouse: int

Чувствительность мыши в пикселях.

ShowDegreeTypeNumeric: bool

Отображать градусы в формате Десятичное значение.

ShowDrawingToolTip: bool

Показывать данные при рисовании.

ShowMapScaleBar: bool

Показывать масштабную линейку.

ShowMeshLayout: bool

Отображать сетку привязки.

ShowMeshLegend: bool

Отображать сетку привязки.

ShowScrollOnMapView: bool

Показывать полосы прокрутки.

ShowSplashScreen: bool

Отображать экран загрузки.

SilentCloseWidget: bool

Подтверждать закрытие несохраненных данных.

SnapSensitiveRadius: int

Привязка узлов - размер

SnapToMeshLayout: bool

Привязывать элементы отчета к сетке.

SnapToMeshLegend: bool

Привязывать к сетке.

SymbolCatalog: Path

Каталог со стилями точек.

UseAntialiasing: bool

Использовать сглаживание при отрисовке.

UseLastSelectedFilter: bool

Запоминать последний фильтр в диалоге открытия файлов.

UseNativeFileDialog: bool

Пользоваться системными файловыми диалогами.

UserDataFolder: Path

Папка с пользовательскими данными Аксиомы.

UserDataPaths: Dict[str, Path]

Список пользовательских каталогов с названиями.

UserPluginsDataFolder: Path

Папка с данными установленных плагинов Аксиомы.

UserPluginsDependenciesFolder: Path

Папка с зависимостями установленных плагинов Аксиомы.

UserPluginsFolder: Path

Корневая папка установленных плагинов Аксиомы.

UserPluginsInstallationFolder: Path

Папка с установленными плагинами Аксиомы.

UserPluginsSettingsFolder: Path

Папка с настройками установленных плагинов Аксиомы.

classmethod get(key: str, default_value: Optional[Any] = None)

Возвращает значение по ключу.

classmethod items() → List[Tuple[str, Any]]

Возвращает список кортежей ключ-значение, где ключи - это атрибуты класса, а значения - это значения настроек.

classmethod keys() → List[str]

Возвращает список ключей, где ключи это атрибуты класса.

static reset()

Сброс значений всех текущих настроек на значения по умолчанию.

classmethod values() → List[Any]

Возвращает список значений, где значения это значения настроек.

class axipy.DefaultSettings

Настройки Аксиомы по умолчанию.

Класс является статическим словарем, доступным только для чтения (`collections.abc.Mapping`). Методы и атрибуты класса такие же как у класса `axipy.CurrentSettings`, но доступны только для чтения.

class axipy.AxiomaLanguage

Язык Аксиомы.

Атрибуты:

en	Английский язык.
ru	Русский язык.

17.6 Вспомогательные функции

`class` ахіру.SelectedFilter

Класс-контейнер, используемый в файловых диалогах, для получения выбранного пользователем фильтра.

filter: `Optional[str]`

Атрибут, который примет значения выбранного фильтра после успешного открытия файлов диалогом.

`ахіру.open_file_dialog`(filter_arg: `Optional[str]` = None, title: `Optional[str]` = None, folder: `Optional[Union[str, Path]]` = None, file_name: `Optional[str]` = None, selected_filter: `Optional[Union[str, SelectedFilter]]` = None, icon: `Optional[QIcon]` = None, set_last_open_path: `bool` = True) → `Optional[Path]`

Открывает диалог выбора файла. Если нет главного окна Аксиомы, спрашивает путь к файлу в консоли.

Параметры

- **filter_arg** – Типы файлов. Например: 'MapInfo Tab (*.tab);; Таблицы Excel (*.xls *.xlsx)'.
- **title** – Заголовок диалога.
- **folder** – Начальная папка диалога. Если папка отсутствует, будет предложена первая из доступных вверх по иерархии или же папка в домашнем каталоге.
- **file_name** – Предлагаемое в диалоге имя файла
- **selected_filter** – Выбранный фильтр типов файлов. Устанавливается необходимая строка из параметра filter_arg. Например, 'Таблицы Excel (*.xls *.xlsx)'. Также, можно использовать класс-контейнер ахіру.SelectedFilter, чтобы получить значение фильтра, выбранное пользователем.
- **icon** – Иконка для диалога, если необходимо ее переопределить
- **set_last_open_path** – Автоматически запоминать путь открытия в настройках Аксиомы.

Результат

Возвращает путь к выбранному файлу или None.

`ахіру.open_files_dialog`(filter_arg: `Optional[str]` = None, title: `Optional[str]` = None, folder: `Optional[Union[str, Path]]` = None, selected_filter: `Optional[Union[str, SelectedFilter]]` = None, icon: `Optional[QIcon]` = None, set_last_open_path: `bool` = True) → `Optional[Tuple[Path, ...]]`

Открывает диалог выбора нескольких файлов.

Параметры

- **filter_arg** – Типы файлов. Например: 'MapInfo Tab (*.tab);; Таблицы Excel (*.xls *.xlsx)'.
- **title** – Заголовок диалога.

- **folder** - Начальная папка диалога или же имя файла. Если папка отсутствует, будет предложена первая из доступных вверх по иерархии или же папка в домашнем каталоге.
- **selected_filter** - Выбранный фильтр типов файлов. Устанавливается необходимая строка из параметра `filter_arg`. Например, 'Таблицы Excel (*.xls *.xlsx)'. Также, можно использовать класс-контейнер `axipy.SelectedFilter`, чтобы получить значение фильтра, выбранное пользователем.
- **icon** - Иконка для диалога, если необходимо ее переопределить
- **set_last_open_path** - Автоматически запоминать путь открытия в настройках Аксиомы.

Результат

Возвращает список с путями к выбранным файлам или None.

Пример:

```
title = 'Файлы для открытия'
filter = 'MapInfo Tab (*.tab);;Таблицы Excel (*.xls *.xlsx)'
folder = Path('/home/user/outfile.tab')
selected_filter = 'MapInfo Tab (*.tab)'
res = axipy.open_files_dialog(filter, title, folder, selected_filter)
[print(fn) for fn in res]
```

```
axipy.save_file_dialog(filter_arg: Optional[str] = None, title: Optional[str] = None,
                      folder: Optional[Path] = None, file_name: Optional[str] = None,
                      selected_filter: Optional[Union[str, SelectedFilter]] = None,
                      icon: Optional[QIcon] = None, set_last_save_path: bool = True)
                      → Optional[Path]
```

Открывает диалог сохранения файла. Если нет главного окна Аксиомы, спрашивает путь к файлу в консоли.

Параметры

- **filter_arg** - Типы файлов. Например: 'MapInfo Tab (*.tab);;Таблицы Excel (*.xls *.xlsx)'.
- **title** - Заголовок диалога.
- **folder** - Начальная папка диалога. Если папка отсутствует, будет предложена первая из доступных вверх по иерархии или же папка в домашнем каталоге.
- **file_name** - Предлагаемое в диалоге имя файла.
- **selected_filter** - Выбранный фильтр типов файлов. Устанавливается необходимая строка из параметра `filter_arg`. Например, 'Таблицы Excel (*.xls *.xlsx)'. Также, можно использовать класс-контейнер `axipy.SelectedFilter`, чтобы получить значение фильтра, выбранное пользователем.
- **icon** - Иконка для диалога, если необходимо ее переопределить.
- **set_last_save_path** - Автоматически запоминать путь сохранения в настройках Аксиомы.

Результат

Возвращает выбранный путь сохранения или None.

`axipy.select_folder_dialog(title: Optional[str] = None, folder: Optional[Union[str, Path]] = None) → Optional[Path]`

Открывает диалог выбора папки. Если нет главного окна Аксиомы, спрашивает путь к файлу в консоли.

Параметры

- **title** – Заголовок диалога.
- **folder** – Начальная папка диалога. Если папка отсутствует, будет предложена первая из доступных вверх по иерархии или же папка в домашнем каталоге.

Результат

Возвращает выбранную папку или None.

`axipy.execfile(path: Union[str, Path])`

Выполняет скрипт на языке python из файла.

Параметры

path – Путь к исполняемому файлу.

`axipy.get_dependencies_folder() → Path`

Возвращает папку, для установки зависимых пакетов.

`axipy.run_in_gui(function: Callable, *args: Any, **kwargs: Any) → Any`

Выполняет переданную функцию в потоке интерфейса.

Это может быть удобно, когда в процессе выполнения длительной фоновой задачи нужно спросить о чем нибудь пользователя отобразив диалог. Также создавать/взаимодействовать с некоторыми объектами можно только из потока интерфейса.

17.7 Менеджеры

`axipy.data_manager = <axipy.DataManager object>`

Менеджер объектов данных (`axipy.DataManager`).

`axipy.view_manager = <axipy.ViewManager object>`

Менеджер содержимого окон (`axipy.ViewManager`).

`axipy.provider_manager = <axipy.ProviderManager object>`

Менеджер провайдеров данных (Открытие/создание объектов данных) (`axipy.ProviderManager`).

`axipy.selection_manager = <axipy.SelectionManager object>`

Менеджер выборки (выделенных объектов) (`axipy.SelectionManager`).

`axipy.task_manager = <axipy.TaskManager object>`

Менеджер пользовательских задач (`axipy.TaskManager`).

`axipy.action_manager = <axipy.ActionManager object>`

Менеджер действий Аксиомы (`axipy.ActionManager`).

`axipy.observer_manager = <axipy.ObserverManager object>`

Менеджер наблюдателей (`axipy.ObserverManager`).

`axipy.plugin_manager = <axipy.PluginManager object>`
 Менеджер плагинов (`axipy.PluginManager`).

17.8 Диалоги

<code>axipy.show_message(text[, title, icon, details])</code>	Показ сообщения.
<code>axipy.show_dialog(text[, title, buttons, ...])</code>	Отображает диалоговое окно с различными опциями и возвращает информацию о кнопке, нажатой пользователем.
<code>axipy.prompt_string(text[, title, ...])</code>	Диалог запроса строкового значения.
<code>axipy.prompt_int(text[, title, value, ...])</code>	Диалог запроса целого значения.
<code>axipy.prompt_float(text[, title, value, ...])</code>	Диалог запроса вещественного значения.
<code>axipy.prompt_item(text[, title, items, ...])</code>	Диалог выбора значения из выпадающего списка.
<code>axipy.select_style_dialog()</code>	Диалог выбора стиля по геометрии в виде функции.
<code>axipy.prompt_rect([rect, unit])</code>	Диалог выбора параметров прямоугольника.
<code>axipy.prompt_coordsystem([cs])</code>	Диалог выбора координатной системы.

17.8.1 Сообщения

`axipy.show_message(text: str, title: str = "", icon: DlgIcon = DlgIcon.NONE, details: str = "")`
 Показ сообщения.

Параметры

- **text** – Текст диалога
- **title** – Заголовок
- **icon** – Иконка
- **details** – Дополнительная информация в виде текста

Пример:

```
show_message('Сообщение', 'Заголовок', icon=DlgIcon.INFORMATION)
```

`axipy.show_dialog(text: str, title: str = "", buttons: DlgButtons = DlgButtons.OK, icon: DlgIcon = DlgIcon.NONE, default_button: Optional[DlgButtons] = None, details: str = "") → DlgButtons`

Отображает диалоговое окно с различными опциями и возвращает информацию о кнопке, нажатой пользователем.

Параметры

- **text** – Текст диалога
- **title** – Заголовок
- **buttons** – Перечень стандартных кнопок диалога,

- **icon** – Иконка
- **default_button** – Кнопка, выбранная по умолчанию. Должна присутствовать в `buttons`
- **details** – Дополнительная информация в виде текста

Результат

Возвращает выбор пользователя в диалоге

Пример:

```
res = show_dialog('Подтвердить действие?', 'Заголовок',
                  icon=DlgIcon.QUESTION,
                  default_button = DlgButtons.CANCEL,
                  buttons = DlgButtons.YES_NO_CANCEL)
if res == DlgButtons.YES:
    print('Yes')
```

17.8.2 Задание значения

`axipy.prompt_string(text: str, title: str = "", multiline: bool = False, value: str = "")` → `Optional[str]`

Диалог запроса строкового значения.

Параметры

- **text** – Текст диалога
- **title** – Заголовок
- **multiline** – Поддержка задания многострочного текста
- **value** – Значение по умолчанию

Пример:

```
res = prompt_string('Введите текст:', multiline=True)
```

`axipy.prompt_int(text: str, title: str = "", value: int = 0, min_value: int = -2147483647, max_value: int = 2147483647, step: int = 1)` → `Optional[int]`

Диалог запроса целого значения.

Параметры

- **text** – Текст диалога
- **title** – Заголовок
- **value** – Значение по умолчанию
- **min_value** – Минимально возможное значение при задании
- **max_value** – Максимально возможное значение при задании
- **step** – Шаг изменения значения посредством мыши

Пример:

```
value = prompt_int('Введите значение (0..100):', min_value=0, max_value=100)
```



```
axipy.prompt_float(text: str, title: str = "", value: float = 0.0, min_value: float =
    -sys.float_info.max, max_value: float = sys.float_info.max, decimals:
    int = 2, step: float = 1) → Optional[float]
```

Диалог запроса вещественного значения.

Параметры

- **text** – Текст диалога
- **title** – Заголовок
- **value** – Значение по умолчанию
- **min_value** – Минимально возможное значение при задании
- **max_value** – Максимально возможное значение при задании
- **decimals** – Цифр после запятой
- **step** – Шаг изменения значения посредством мыши

Пример:

```
value = prompt_float('Введите значение (0..100):', min_value=0, max_value=100)
```

```
axipy.prompt_item(text: str, title: str = "", items: Optional[Iterable[str]] = None, value:
    Union[int, str] = 0, editable: bool = False) → Optional[str]
```

Диалог выбора значения из выпадающего списка.

Параметры

- **text** – Текст диалога.
- **title** – Заголовок.
- **items** – Последовательность значений или итератор.
- **value** – Значение по умолчанию или его индекс.
- **editable** – Допустимо ли редактирование текущего значения.

Результат

Выбранное значение или пустая строка.

Пример:

```
res = prompt_item('Варианты для выбора:', items = ("один", "два", "три"), value =
    ↪ 'два')
```

```
axipy.select_style_dialog(value: StyleGeometryType) → Optional[Style]
```

```
axipy.select_style_dialog(value: Style) → Optional[Style]
```

```
axipy.select_style_dialog(value: Geometry) → Optional[Style]
```

Диалог выбора стиля по геометрии в виде функции. В качестве входных параметров рассматривается или стиль или геометрия, для которого этот стиль выбирается.

Параметры

value – Геометрия, ее тип или стиль. Если задана геометрия, то стиль по умолчанию подставляется как стиль нового объекта для данного типа геометрии.

Результат

Возвращает выбранный в диалоге стиль.

По типу геометрического объекта:

```
style = axipy.select_style_dialog(StyleGeometryType.Linear)
```

Пример задания по геометрии:

```
style = axipy.select_style_dialog(Point(3,3))
```

Пример задания по стилю:

```
style = select_style_dialog(axipy.Style.from_mapinfo('Symbol (35, 0, 12)'))
```

`axipy.prompt_rect`(rect: `Rect` = `Rect`(-10000000, -10000000, 10000000, 10000000), unit: `Union[LinearUnit, str]` = `LinearUnits.m`) → `Optional[Rect]`

Диалог выбора параметров прямоугольника.

Параметры

- **rect** – Значение по умолчанию.
- **unit** – Единицы измерения при отображении.

См.также:

`axipy.BoundingRectDialog`

`axipy.prompt_coordsystem`(cs: `Optional[CoordSystem]` = `None`) → `Optional[CoordSystem]`

Диалог выбора координатной системы.

Параметры

cs – Система координат.

См.также:

`axipy.ChooseCoordSystemDialog`

axipy.app - Модуль приложения.

Модуль приложения.

Данный модуль является основным модулем приложения.

18.1 MainWindow - Главное окно

`axipy.mainwindow = <axipy.MainWindow object>`

Готовый экземпляр главного окна ГИС Аксиома. (`axipy.MainWindow`).

class `axipy.MainWindow`

Главное окно ГИС Аксиома.

Примечание: Используйте готовый объект `axipy.mainwindow`.

Классовые методы:

<code>show()</code>	Создает и показывает главное окно программы.

Свойства:

<code>dock_widgets</code>	Возвращает список панелей.
<code>geometry</code>	Устанавливает или возвращает положение главного окна.
<code>is_valid</code>	Возвращает признак корректности состояния главного окна.
<code>quick_toolbar</code>	Возвращает панель быстрого запуска.
<code>widget</code>	Возвращает виджет, соответствующий содержимому окна.

Методы:

<code>add(view)</code>	Добавляет окно просмотра данных.
<code>add_dock_widget(dock_widget, area[, icon])</code>	Добавляет панель в главное окно приложения.
<code>add_layer_current_map(layer)</code>	Добавляет слой в текущей карте.
<code>add_layer_interactive(layer)</code>	Добавляет слой с запросом на помещение на текущую карту или в новую.
<code>add_layer_new_map(layer)</code>	Открывает слой в новой карте.
<code>dock_widget_by_name(name)</code>	Производит поиск панелей по заголовку.
<code>remove_dock_widget(dock)</code>	Удаляет существующую панель у главного окна приложения.
<code>show_html_url(url, caption)</code>	Показывает окно для локального файла html или если это web страница, запускает браузер по ассоциации.

add(view: `View`) → `QMdiSubWindow`

Добавляет окно просмотра данных.

Параметры

view – окно просмотра данных.

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

add_dock_widget(dock_widget: `QDockWidget`, area: `DockWidgetArea`, icon: `Optional[QIcon]` = None) → `bool`

Добавляет панель в главное окно приложения. При успешном добавлении возвращает True. Если же данная панель уже присутствует, то команда игнорируется и возвращается False. Элементы управления, которые требуется разместить на панели, создаются в дополнительном окне, а уже это окно, в свою очередь, устанавливается для панели (см. пример ниже).

Параметры

- **dock_widget** – Пользовательская созданная панель.
- **area** – Расположение.
- **icon** – Иконка для отображения в списке всех доступных панелей.

Пример:

```
from PySide2.QtWidgets import QDockWidget, QWidget, QPushButton
from PySide2.QtCore import Qt

dock = QDockWidget('Заголовок')
widget = QWidget()
layout = QVBoxLayout()
button = QPushButton("Кнопка")
button.clicked.connect(lambda: print('Реакция на кнопку'))
layout.addWidget(button)
layout.addStretch()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
widget.setLayout(layout)
dock.setWidget(widget)
app.mainwindow.add_dock_widget(dock, Qt.RightDockWidgetArea, QIcon('filename.
→png'))
```

add_layer_current_map(layer: [Layer](#)) → [MapView](#)

Добавляет слой в текущей карте.

add_layer_interactive(layer: [Layer](#)) → [MapView](#)

Добавляет слой с запросом на помещение на текущую карту или в новую.

add_layer_new_map(layer: [Layer](#)) → [MapView](#)

Открывает слой в новой карте.

dock_widget_by_name(name: [str](#)) → [Tuple\[QDockWidget, ...\]](#)

Производит поиск панелей по заголовку.

Параметры

name – Заголовок панели или же шаблон в формате [re](#)

Пример выдачи всех окон, у которых заголовок начинается с «Заголовок»

```
for dock in app.mainwindow.dock_widget_by_name('Заголовок.*$'):
    print(dock.windowTitle())

>>> Заголовок2
>>> Заголовок
>>> Заголовок1
```

property dock_widgets: [Tuple\[QDockWidget, ...\]](#)

Возвращает список панелей.

property geometry: [QRect](#)

Устанавливает или возвращает положение главного окна.

property is_valid: [bool](#)

Возвращает признак корректности состояния главного окна.

property quick_toolbar: [Optional\[QToolBar\]](#)

Возвращает панель быстрого запуска. Доступна только для ленточного интерфейса.

Пример добавления инструмента выбора:

```
tool_bar = mainwindow.quick_toolbar
if tool_bar:
    tool_bar.addAction(action_manager['Select'])
```

remove_dock_widget(dock: [QDockWidget](#))

Удаляет существующую панель у главного окна приложения.

static show() → [MainWindow](#)

Создает и показывает главное окно программы.

show_html_url(url: [QUrl](#), caption: [str](#))

Показывает окно для локального файла html или если это web страница, запускает браузер по ассоциации.

Параметры

- **url** – Ссылка на файл html или адрес страницы.
- **caption** – Заголовок окна.

property widget: **Optional[QMainWindow]**

Возвращает виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

class ахіру.**DockWidgetArea**

Расположение панели.

Атрибуты:

Bottom	Снизу.
Left	Слева.
Right	Справа.
Top	Сверху.

18.2 Version - Информация о версии

class ахіру.**Version**

Информация о версии ГИС Аксиома.

Пример использования:

```
from ахіру import Version

# В данном случае, если текущая версия выше 4.3 (например 5.1), то возвращается -
→ 1.
print('Версия:', Version.string(), Version.compare(4,3))
```

Классовые методы:

<code>compare(major[, minor, patch])</code>	Сравнивает переданное значение с текущей версией Аксиомы.
<code>full_string()</code>	Возвращает полную версию в виде строки, включая исправления patch.
<code>number()</code>	Возвращает версию в виде одного числа, в котором каждый сегмент располагается в отдельном байте.
<code>qtFormat()</code>	Возвращает версию в Qt формате.
<code>qt_format()</code>	Возвращает версию в Qt формате.
<code>segments()</code>	Возвращает кортеж чисел - сегменты версии: (major, minor, patch).
<code>string()</code>	Возвращает версию в виде строки.

static `compare(major: int, minor: int = 0, patch: int = 0) → int`

Сравнивает переданное значение с текущей версией Аксиомы.

Параметры

- **major** - Основная версия.
- **minor** - Минорная версия.
- **patch** - Исправления.

Возвращает -1, если переданная меньше, 1 если больше и 0 если равны.

static full_string() → **str**

Возвращает полную версию в виде строки, включая исправления patch.

static number() → **int**

Возвращает версию в виде одного числа, в котором каждый сегмент располагается в отдельном байте.

static qtFormat() → **QVersionNumber**

Возвращает версию в Qt формате. :meta private:

static qt_format() → **QVersionNumber**

Возвращает версию в Qt формате.

static segments() → **tuple**

Возвращает кортеж чисел - сегменты версии: (major, minor, patch).

static string() → **str**

Возвращает версию в виде строки.

axipy.cs - Модуль систем координат.

Модуль систем координат.

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

19.1 CoordSystem - Система Координат (СК)

class axipy.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.

Примечание: Проверка на идентичность параметров двух СК производится простым сравнением.

Примечание: Для получения текстового представления можно воспользоваться функцией `str`.

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()`

Список 1: Пример создания СК разного типа.

```
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.
↪ 017453292519943295]]')
# Создание из строки с указанием вида формата
crs1 = CoordSystem.from_string('epsg:4326')
crs2 = CoordSystem.from_string('prj:1,104')
```

Список 2: Проверка на идентичность координатных систем производится простым сравнением.

```
cs1 = CoordSystem.from_prj("1, 104")
cs2 = CoordSystem.from_prj("1, 104")
if cs1 == cs2:
    print("Координатные системы эквивалентны.")
...
>>> Координатные системы эквивалентны.
...
```

Классовые методы:

<code>current()</code>	Текущая установленная система координат (СК).
<code>from_epsg(code)</code>	Создает координатную систему по коду EPSG.
<code>from_prj(prj)</code>	Создает координатную систему из строки MapBasic.
<code>from_proj(proj)</code>	Создает координатную систему из строки proj.
<code>from_string(string)</code>	Создает систему координат из строки.
<code>from_units(unit[, rect])</code>	Создает декартову систему координат.
<code>from_wkt(wkt)</code>	Создает координатную систему из строки WKT.
<code>set_current(coordsystem)</code>	Устанавливает новую текущую систему координат.

Свойства:

<code>default_calc_mode</code>	Возвращает режим расчета для данной СК по умолчанию.
<code>epsg</code>	Возвращает код EPSG если существует, иначе None.
<code>inv_flattening</code>	Возвращает полярное сжатие.
<code>lat_lon</code>	Возвращает признак, является ли СК широтой/долготой.
<code>name</code>	Возвращает наименование системы координат.
<code>non_earth</code>	Возвращает признак, является ли СК декартовой.
<code>prj</code>	Возвращает строку prj формата MapBasic или пустую строку, если аналога не найдено.
<code>proj</code>	Возвращает строку PROJ или пустую строку, если аналога не найдено.
<code>rect</code>	Устанавливает или возвращает максимально допустимый охват.
<code>semi_major</code>	Возвращает большую полуось.
<code>semi_minor</code>	Возвращает малую полуось.
<code>supported_calc_modes</code>	Возвращает возможный перечень режимов расчета для данной СК.
<code>title</code>	Возвращает наименование системы координат.
<code>unit</code>	Возвращает единицы измерения.
<code>wkt</code>	Возвращает строку WKT или пустую строку, если аналога не найдено.

Методы:

<code>clone()</code>	Возвращает копию координатной системы.
<code>convert_from_degree()</code>	Переводит из градусов в единицы измерения системы координат.
<code>convert_to_degree()</code>	Переводит из единиц измерения системы координат в градусы.
<code>is_equal(other[, ignore_rect])</code>	Сравнивает координатные системы, с возможностью не учитывать охват.
<code>to_string()</code>	Текстовое представление в виде <тип>:<строка>.

Специальные методы:

<code>__copy__()</code>	Поверхностная копия объекта.
<code>__deepcopy__(_mem_dict)</code>	Глубокая копия объекта.
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

__copy__() → [CoordSystem](#)

Поверхностная копия объекта. (функция [copy.copy\(\)](#))

__deepcopy__(_mem_dict: dict) → [CoordSystem](#)

Глубокая копия объекта. (функция [copy.deepcopy\(\)](#))

__eq__(other: object) → [bool](#)

Оператор равенства (==).

__repr__() → [str](#)

Официальное строковое представление объекта при вызове [repr\(\)](#).

__str__() → [str](#)

Неформальное строковое представление объекта при вызове [str](#) или [print\(\)](#).

clone() → [CoordSystem](#)

Возвращает копию координатной системы.

convert_from_degree(value: Union[Rect, QRectF]) → [Rect](#)

convert_from_degree(value: Union[Tuple[float, float], Pnt, QPointF]) → [Pnt](#)

convert_from_degree(value: List[Union[Tuple[float, float], Pnt, QPointF]]) → [List\[Pnt\]](#)

Переводит из градусов в единицы измерения системы координат.

convert_to_degree(value: Union[Rect, QRectF]) → [Rect](#)

convert_to_degree(value: Union[Tuple[float, float], Pnt, QPointF]) → [Pnt](#)

convert_to_degree(value: List[Union[Tuple[float, float], Pnt, QPointF]]) → [List\[Pnt\]](#)

Переводит из единиц измерения системы координат в градусы.

Список 3: Пример.

```
csMercator = CoordSystem.from_prj("10, 104, 7, 0")
p_out = csMercator.convert_to_degree((1000000, 1000000))
print(p_out)
...
>>> (8.983152841195214 9.005882635078796)
...
```

classmethod current() → [CoordSystem](#)

Текущая установленная система координат (СК).

Данная СК используется как значение по умолчанию, когда она не определена.
Например, в диалоге создания новой таблицы.

property default_calc_mode: [CalcMode](#)

Возвращает режим расчета для данной СК по умолчанию.

property epsg: [Optional\[int\]](#)

Возвращает код EPSG если существует, иначе None.

classmethod from_epsg(code: int) → [CoordSystem](#)

Создает координатную систему по коду EPSG.

См.также:

Подробнее см. [EPSG](#)

Параметры

code – Стандартное значение EPSG.

classmethod from_prj(prj: str) → CoordSystem

Создает координатную систему из строки MapBasic.

Параметры

prj – Строка MapBasic. Допустима короткая нотация.

Список 4: Пример.

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

classmethod from_proj(proj: str) → CoordSystem

Создает координатную систему из строки proj.

См.также:

Подробнее см. [PROJ](#)

Параметры

proj – Строка proj.

classmethod from_string(string: str) → CoordSystem

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

Параметры

string – Строка.

classmethod from_units(unit: LinearUnit, rect: Optional[Union[Rect, QRectF]] = None) → CoordSystem

Создает декартову систему координат.

Параметры

- **unit** – Единицы измерения системы координат.
- **rect** – Охват системы координат. По умолчанию, ахіру. Rect(xmin=-10000, ymin=-10000, xmax=10000, ymax=10000).

Список 5: Пример.

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

classmethod from_wkt(wkt: str) → CoordSystem

Создает координатную систему из строки WKT.

См.также:

Подробнее см.

[WKT](#)

Параметры

wkt – Строка WKT.

property inv_flattening: float

Возвращает полярное сжатие.

is_equal(other: [CoordSystem](#), ignore_rect: [bool](#) = False) → [bool](#)

Сравнивает координатные системы, с возможностью не учитывать охват.

property lat_lon: [bool](#)

Возвращает признак, является ли СК широтой/долготой.

property name: [str](#)

Возвращает наименование системы координат.

property non_earth: [bool](#)

Возвращает признак, является ли СК декартовой.

property prj: [str](#)

Возвращает строку prj формата MapBasic или пустую строку, если аналога не найдено.

property proj: [str](#)

Возвращает строку PROJ или пустую строку, если аналога не найдено.

property rect: [Rect](#)

Устанавливает или возвращает максимально допустимый охват.

property semi_major: [float](#)

Возвращает большую полуось.

property semi_minor: [float](#)

Возвращает малую полуось.

classmethod set_current(coordsystem: [CoordSystem](#))

Устанавливает новую текущую систему координат.

Параметры

coordsystem – Новое значение системы координат.

Пример установки нового значения:

```
CoordSystem.set_current(CoordSystem.from_prj("10, 104, 7"))
```

property supported_calc_modes: [Tuple](#)[[CalcMode](#), ...]

Возвращает возможный перечень режимов расчета для данной СК.

property title: [str](#)

Возвращает наименование системы координат.

to_string() → [str](#)

Текстовое представление в виде <тип>:<строка>.

property unit: [LinearUnit](#)

Возвращает единицы измерения.

property wkt: [str](#)

Возвращает строку WKT или пустую строку, если аналога не найдено.

19.2 CoordTransformer - Трансформация координат

class axipy.CoordTransformer

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.

Список 6: Пример преобразования точки

```
from axipy import CoordSystem, CoordTransformer, Pnt, Rect

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)
print('Result point:', outPoint)
'''
>>> Result point: (1113194.9079327357 1111475.1028522244)
'''

outRect = transformer.transform(Rect(0,0,10,10))
print('Result rect:', outRect)
'''
>>> Result rect: (0.0 0.0) (1113194.9079327357 1111475.1028522244)
'''
```

Конструктор класса:

<code>__init__(cs_from, cs_to)</code>	Конструктор класса.

Классовые методы:

<code>proj_transform_definition(cs_from, cs_to)</code>	Возвращает строку трансформации (pipeline) для преобразования между двумя СК, заданными в формате proj.

Методы:

<code>transform()</code>	Преобразовывает точки из исходной СК в целевую СК.

`__init__(cs_from: CoordSystem, cs_to: CoordSystem)`

Конструктор класса.

Параметры

- `cs_from` - Исходная СК.
- `cs_to` - Целевая СК.

`classmethod proj_transform_definition(cs_from: str, cs_to: str) → str`

Возвращает строку трансформации (pipeline) для преобразования между двумя СК, заданными в формате proj.

Параметры

- **cs_from** - Строка с определением исходной СК в формате proj
- **cs_to** - Строка с определением исходной СК в формате proj

Результат

Строка с определением трансформации между двумя этими СК в формате proj

Список 7: Пример получения строки трансформации

```
str_from = '+proj=longlat +ellps=WGS84 +no_defs'
str_to = '+proj=merc +ellps=GRS80 +no_defs'
print(CoordTransformer.proj_transform_definition(str_from, str_to))
'''
>>> proj=pipeline step proj=unitconvert xy_in=deg xy_out=rad step proj=merc
lon_0=0 k=1 x_0=0 y_0=0 ellps=GRS80
'''
```

transform(value: Union[QRectF, Rect]) → Rect

transform(value: Union[Tuple[float, float], Pnt, QPointF]) → Pnt

transform(value: Iterable[Union[Tuple[float, float], Pnt, QPointF]]) → List[Pnt]

Преобразовывает точки из исходной СК в целевую СК.

Параметры

value - Входное значение. Может быть точкой, массивом точек `axipy.Pnt` или `axipy.Rect`.

Результат

Выходное значение. Тип зависит от входного и аналогичен ему.

Исключение

RuntimeError - Ошибка выполнения преобразования.

19.3 LinearUnits - Единицы измерения расстояний

class `axipy.LinearUnits`

Единицы измерения расстояний.

Класс является статическим словарем, доступным только для чтения (`collections.abc.Mapping`). Поддерживает обращение по индексу.

Классовые методы:

<code>get(key[, default_value])</code>	Возвращает значение по ключу.
<code>items()</code>	Возвращает список кортежей ключ-значение, где ключи это атрибуты класса, а значения это объекты класса <code>axipy.LinearUnit</code> .
<code>keys()</code>	Возвращает список ключей, где ключи это атрибуты класса.
<code>values()</code>	Возвращает список значений, где значения это объекты класса <code>axipy.LinearUnit</code> .

Атрибуты:

ch	Чейны.
cm	Сантиметры.
degree	Градусы.
ft	Футы.
inch	Дюймы.
km	Километры.
li	Линки.
m	Метры.
mi	Мили.
mm	Миллиметры.
nmi	Морские мили.
rd	Роды.
survey_ft	Топографические футы.
yd	Ярды.

class axipy.LinearUnit

Линейная единица измерения.

Используется для работы с координатами объектов или расстояний.

Примечание: Получить экземпляр можно через класс `axipy.LinearUnits` по соответствующему атрибуту.

Список 8: Пример создания

```
meters = LinearUnits.m # LinearUnit
square_kilometers = AreaUnits.sq_km # AreaUnit
```

Классовые методы:

<code>from_area_unit(area_unit)</code>	Возвращает единицу измерения расстояния, соответствующую единице измерения площадей.

Свойства:

<code>conversion</code>	Коэффициент преобразования в метры.
<code>description</code>	Краткое описание.
<code>localized_name</code>	Локализованное краткое наименование единиц измерения.
<code>name</code>	Краткое наименование единиц измерения.

Методы:

<code>to_unit(unit[, value])</code>	Перевод значения в другие единицы измерения.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property conversion: float

Коэффициент преобразования в метры.

property description: str

Краткое описание.

static from_area_unit(area_unit: AreaUnit) → Optional[LinearUnit]

Возвращает единицу измерения расстояния, соответствующую единице измерения площадей.

property localized_name: str

Локализованное краткое наименование единиц измерения.

property name: str

Краткое наименование единиц измерения.

to_unit(unit: Unit, value: float = 1) → float

Перевод значения в другие единицы измерения.

Параметры

- **unit** – Единицы измерения, в которые необходимо перевести значение.
- **value** – Значение для перевода.

19.4 AreaUnits - Единицы измерения площадей

class ахіру.AreaUnits

Единицы измерения площадей.

Класс является статическим словарем, доступным только для чтения (`collections.abc.Mapping`). Поддерживает обращение по индексу.

Классовые методы:

<code>get(key[, default_value])</code>	Возвращает значение по ключу.
<code>items()</code>	Возвращает список кортежей ключ-значение, где ключи это атрибуты класса, а значения это объекты класса <code>axipy.AreaUnit</code> .
<code>keys()</code>	Возвращает список ключей, где ключи это атрибуты класса.
<code>values()</code>	Возвращает список значений, где значения это объекты класса <code>axipy.AreaUnit</code> .

Атрибуты:

<code>acre</code>	Акры.
<code>hectare</code>	Гектары.
<code>perch</code>	Перчи.
<code>rood</code>	Руды.
<code>sq_ch</code>	Квадратные чейны.
<code>sq_cm</code>	Квадратные сантиметры.
<code>sq_ft</code>	Квадратные футы.
<code>sq_inch</code>	Квадратные дюймы.
<code>sq_km</code>	Квадратные километры.
<code>sq_li</code>	Квадратные линки.
<code>sq_m</code>	Квадратные метры.
<code>sq_mi</code>	Квадратные мили.
<code>sq_mm</code>	Квадратные миллиметры.
<code>sq_nmi</code>	Квадратные морские мили.
<code>sq_rd</code>	Квадратные роды.
<code>sq_survey_ft</code>	Квадратные топографические футы.
<code>sq_yd</code>	Квадратные ярды.

`class axipy.AreaUnit`

Единица измерения площадей.

Примечание: Получить экземпляр можно через класс `axipy.AreaUnits` по соответствующему атрибуту.

Список 9: Пример создания

```
meters = LinearUnits.m # LinearUnit
square_kilometers = AreaUnits.sq_km # AreaUnit
```

Классовые методы:

<code>from_linear_unit(linear_unit)</code>	Возвращает единицу измерения площадей, соответствующую единице измерения расстояний.

Свойства:

<code>conversion</code>	Коефіцієнт преобразования в метры.
<code>description</code>	Краткое описание.
<code>localized_name</code>	Локализованное краткое наименование единиц измерения.
<code>name</code>	Краткое наименование единиц измерения.

Методы:

<code>to_unit(unit[, value])</code>	Перевод значения в другие единицы измерения.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property conversion: float

Коефіцієнт преобразования в метры.

property description: str

Краткое описание.

static from_linear_unit(linear_unit: LinearUnit) → Optional[AreaUnit]

Возвращает единицу измерения площадей, соответствующую единице измерения расстояний.

property localized_name: str

Локализованное краткое наименование единиц измерения.

property name: str

Краткое наименование единиц измерения.

to_unit(unit: Unit, value: float = 1) → float

Перевод значения в другие единицы измерения.

Параметры

- **unit** – Единицы измерения, в которые необходимо перевести значение.
- **value** – Значение для перевода.

19.5 UnitValue - Значение вместе с единицей измерения

class axipy.UnitValue

Контейнер, который хранит значение вместе с его единицей измерения.

Пример:

```
unit = axipy.UnitValue(2, axipy.LinearUnits.km)
print(unit)
>>> "2 km"
```

Конструктор класса:

<code>__init__</code> ([value, unit])	Конструктор класса.

Свойства:

<code>unit</code>	Устанавливает или возвращает единицу измерения.
<code>value</code>	Устанавливает или возвращает значение.

Специальные методы:

<code>__eq__</code> (other)	Оператор равенства (==).
<code>__ge__</code> (other)	Оператор 'Больше или равно' (>=).
<code>__gt__</code> (other)	Оператор 'Больше' (>).
<code>__le__</code> (other)	Оператор 'Меньше или равно' (<=).
<code>__lt__</code> (other)	Оператор 'Меньше' (<).
<code>__repr__</code> ()	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__</code> ()	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__`(other: object) → bool

Оператор равенства (==).

`__ge__`(other: UnitValue) → bool

Оператор „Больше или равно“ (>=).

`__gt__`(other: UnitValue) → bool

Оператор „Больше“ (>).

`__init__`(value: float = 1, unit: Optional[Unit] = LinearUnits.m)

Конструктор класса.

Параметры

- **value** – Значение.
- **unit** – Единица измерения, в которой содержится значение. Если значение не указано, принимаются метры `LinearUnits.m`.

`__le__(other: UnitValue) → bool`

Оператор „Меньше или равно“ (`<=`).

`__lt__(other: UnitValue) → bool`

Оператор „Меньше“ (`<`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property unit: Unit

Устанавливает или возвращает единицу измерения.

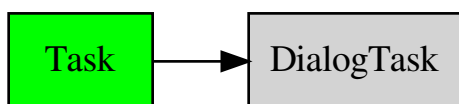
property value: float

Устанавливает или возвращает значение.

axipy.concurrent - Модуль для работы с длительными задачами в фоновом потоке.

Модуль для работы с длительными пользовательскими задачами, выполняемыми в фоновом потоке.

20.1 Task - Пользовательская задача



class axipy.Task

Пользовательская задача. Если выполнение функции в программе занимает длительное время, графический интерфейс Аксиомы будет неотзывчивым в течении этого времени. Чтобы длительная задача выполнялось в отдельном потоке и не нарушала работу интерфейса Аксиомы, такую функцию следует поместить в задачу, используя класс `axipy.Task`.

Задачи (объекты класса `axipy.Task`) добавляются в менеджер `axipy.task_manager` при создании конструктором, и удаляются при завершении задачи (успешно или с ошибкой). Еще не запущенную задачу, можно удалить из менеджера, вызвав метод `axipy.Task.cancel`. Одну задачу можно запустить только один раз, завершенную задачу нельзя запустить повторно.

См.также:

`axipy.run_in_gui()`

См.также:

Работа с длительными операциями

Конструктор класса:

<code>__init__</code> (function[, name, description, ...])	Конструктор класса.

Свойства:

<code>description</code>	Возвращает описание задачи.
<code>id</code>	Возвращает идентификатор задачи.
<code>is_canceled</code>	Возвращает True, если задача была отменена.
<code>max</code>	Устанавливает или возвращает максимальное значение диапазона прогресса задачи.
<code>message</code>	Устанавливает или возвращает отображаемое описание задачи.
<code>min</code>	Устанавливает или возвращает минимальное значение диапазона прогресса задачи.
<code>name</code>	Возвращает имя задачи.
<code>range</code>	Устанавливает или возвращает диапазон прогресса задачи.
<code>result</code>	Возвращает результат задачи.
<code>status</code>	Возвращает текущее состояние задачи.
<code>thread_id</code>	Возвращает идентификатор потока, в котором выполняется задача.
<code>title</code>	Устанавливает или возвращает отображаемое имя задачи.
<code>value</code>	Устанавливает или возвращает текущее значение прогресса задачи.

Методы:

<code>cancel()</code>	Уведомляет задачу об отмене.
<code>raise_if_canceled([message])</code>	Вызывает исключение об отмене задачи <code>axipy.Task.CanceledException</code> .
<code>run_and_get(*args, **kwargs)</code>	Запускает задачу и ждет результат.
<code>start(*args, **kwargs)</code>	Запускает задачу без ожидания результата.

Сигналы:

<code>finished</code>	Возвращает сигнал, испускаемый при завершении задачи.
<code>message_changed</code>	Возвращает сигнал, испускаемый при изменении отображаемого описании задачи.
<code>range_changed</code>	Возвращает сигнал, испускаемый при изменении диапазона прогресса задачи.
<code>started</code>	Возвращает сигнал, испускаемый при старте задачи.
<code>title_changed</code>	Возвращает сигнал, испускаемый при изменении отображаемого имени задачи.
<code>value_changed</code>	Возвращает сигнал, испускаемый при изменении текущего значения прогресса задачи.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

Исключения:

<code>CanceledException</code>	Исключение при отмене задачи.

Классы:

<code>Range</code>	Диапазон прогресса задачи.
<code>Status</code>	Состояние задачи.

exception CanceledException

Исключение при отмене задачи.

class Range

Диапазон прогресса задачи.

Атрибуты:

<code>min</code>	Минимальное значение прогресса задачи.
<code>max</code>	Максимальное значение прогресса задачи.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

class Status

Состояние задачи.

Атрибуты:

IDLE	В ожидании.
RUNNING	Запущена.
SUCCESS	Выполнена успешно.
CANCELED	Отменена.
ERROR	Завершилась с ошибкой.

__init__ (function: Callable, name: str = "", description: str = "", title: str = "", message: str = "")

Конструктор класса.

Создает экземпляр класса.

Параметры

- **function** – Функция.
- **name** – Имя.
- **description** – Описание.
- **title** – Отображаемое имя.
- **message** – Отображаемое описание.

__repr__() → str

Официальное строковое представление объекта при вызове `repr()`.

__str__() → str

Неформальное строковое представление объекта при вызове `str` или `print()`.

cancel()

Уведомляет задачу об отмене.

Устанавливает свойство `axipy.Task.is_canceled` на True и меняет состояние задачи `axipy.Task.status` на `axipy.Task.Status.CANCELED`. Вызов метода при статусе `axipy.Task.Status.IDLE` также удалит задачу из `axipy.task_manager`. Метод является слотом, Slot() (PySide2.QtCore.Slot).

property description: str

Возвращает описание задачи.

property finished: Signal

Возвращает сигнал, испускаемый при завершении задачи. Передает ссылку на экземпляр задачи, для получения результата, статуса и других атрибутов задачи.

Тип результата

Signal[axipy.Task]

property id: int

Возвращает идентификатор задачи.

Идентификатором задачи является её порядковый номер в менеджере задач `axipy.task_manager`.

property is_canceled: bool

Возвращает True, если задача была отменена.

Чтобы отменить задачу (назначить True для свойства `axipy.Task.is_canceled`), нужно вызвать `axipy.Task.cancel()`.

property max: float

Устанавливает или возвращает максимальное значение диапазона прогресса задачи.

property message: str

Устанавливает или возвращает отображаемое описание задачи.

property message_changed: Signal

Возвращает сигнал, испускаемый при изменении отображаемого описании задачи.

Тип результата

Signal[str]

property min: float

Устанавливает или возвращает минимальное значение диапазона прогресса задачи.

property name: str

Возвращает имя задачи.

raise_if_canceled(message: Optional[str] = None)

Вызывает исключение об отмене задачи `axipy.Task.CanceledException`.

Параметры

message – Сообщение, передаваемое в исключение.

property range: Range

Устанавливает или возвращает диапазон прогресса задачи.

property range_changed: Signal

Возвращает сигнал, испускаемый при изменении диапазона прогресса задачи.

Тип результата

Signal[axipy.Task.Range]

property result: Any

Возвращает результат задачи. (Возвращаемое значение пользовательской функции.) Если во время выполнения задачи, было вызвано исключение, то при получении результата, исключение будет вызвано повторно (За исключением пользовательской отмены `axipy.Task.CanceledException`).

Исключение

Exception – Если во время выполнения задачи, было вызвано исключение.

run_and_get(*args: Any, **kwargs: Any) → Any

Запускает задачу и ждет результат. Останавливается лишь выполнение кода python, цикл обработки событий Qt продолжает работу. (Сигналы и события продолжают обрабатываться.) Возвращает результат задачи. (Возвращаемое значение пользовательской функции.) Если во время выполнения задачи, было вызвано исключение, то при получении результата, исключение будет

вызвано повторно (За исключением пользовательской отмены `axipy.Task.CanceledException`).

Параметры

- ***args** – Параметры, передаваемые в пользовательскую функцию.
- ****kwargs** – Именованные параметры, передаваемые в пользовательскую функцию.

Результат

Результат выполнения задачи. (Возвращаемое значение пользовательской функции.)

Исключение

Exception – Если во время выполнения задачи, было вызвано исключение.

start(*args: `Any`, **kwargs: `Any`)

Запускает задачу без ожидания результата. Чтобы получить результат, необходимо подписаться на сигнал `axipy.Task.finished` и обратиться к свойству `axipy.Task.result`.

Параметры

- ***args** – Параметры, передаваемые в пользовательскую функцию.
- ****kwargs** – Именованные параметры, передаваемые в пользовательскую функцию.

property started: `Signal`

Возвращает сигнал, испускаемый при старте задачи.

Тип результата

`Signal[]`

property status: `Status`

Возвращает текущее состояние задачи.

property thread_id: `Optional[int]`

Возвращает идентификатор потока, в котором выполняется задача.

Когда задача не выполняется, возвращает `None`.

property title: `str`

Устанавливает или возвращает отображаемое имя задачи.

property title_changed: `Signal`

Возвращает сигнал, испускаемый при изменении отображаемого имени задачи.

Тип результата

`Signal[str]`

property value: `float`

Устанавливает или возвращает текущее значение прогресса задачи.

property value_changed: `Signal`

Возвращает сигнал, испускаемый при изменении текущего значения прогресса задачи.

Тип результата
Signal[[str](#)]

20.2 DialogTask - Пользовательская задача с диалогом

class `axipy.DialogTask`

Базовые классы: [Task](#)

Пользовательская задача с диалогом. Наследуется от класса `axipy.Task`.
Дополняет класс `axipy.Task` диалогом, показывающим прогресс задачи.

См.также:

`axipy.run_in_gui()`

См.также:

Работа с длительными операциями

Конструктор класса:

```
__init__(function[, name, description, Конструктор класса.  
...])
```

Свойства:

<code>cancelable</code>	Возвращает True, если в диалоге есть кнопка отмены.
<code>delayed</code>	Возвращает True, если в диалоге есть задержка перед показом.
<code>description</code>	Возвращает описание задачи.
<code>id</code>	Возвращает идентификатор задачи.
<code>infinite_progress</code>	Возвращает или устанавливает бесконечную полосу прогресса для задачи.
<code>is_canceled</code>	Возвращает True, если задача была отменена.
<code>max</code>	Устанавливает или возвращает максимальное значение диапазона прогресса задачи.
<code>message</code>	Устанавливает или возвращает отображаемое описание задачи.
<code>min</code>	Устанавливает или возвращает минимальное значение диапазона прогресса задачи.
<code>name</code>	Возвращает имя задачи.
<code>range</code>	Устанавливает или возвращает диапазон прогресса задачи.
<code>result</code>	Возвращает результат задачи.
<code>status</code>	Возвращает текущее состояние задачи.
<code>thread_id</code>	Возвращает идентификатор потока, в котором выполняется задача.
<code>title</code>	Устанавливает или возвращает отображаемое имя задачи.
<code>value</code>	Устанавливает или возвращает текущее значение прогресса задачи.

Методы:

<code>cancel()</code>	Уведомляет задачу об отмене.
<code>raise_if_canceled([message])</code>	Вызывает исключение об отмене задачи <code>axipy.Task.CanceledException</code> .
<code>run_and_get(*args, **kwargs)</code>	Запускает задачу и ждет результат.
<code>start(*args, **kwargs)</code>	Запускает задачу без ожидания результата.

Сигналы:

<code>finished</code>	Возвращает сигнал, испускаемый при завершении задачи.
<code>message_changed</code>	Возвращает сигнал, испускаемый при изменении отображаемого описании задачи.
<code>range_changed</code>	Возвращает сигнал, испускаемый при изменении диапазона прогресса задачи.
<code>started</code>	Возвращает сигнал, испускаемый при старте задачи.
<code>title_changed</code>	Возвращает сигнал, испускаемый при изменении отображаемого имени задачи.
<code>value_changed</code>	Возвращает сигнал, испускаемый при изменении текущего значения прогресса задачи.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

Исключения:

<code>CanceledException</code>	Исключение при отмене задачи.

Классы:

<code>Range</code>	Диапазон прогресса задачи.
<code>Status</code>	Состояние задачи.

exception CanceledException

Базовые классы: `Exception`

Исключение при отмене задачи.

class Range

Базовые классы: `tuple`

Диапазон прогресса задачи.

Атрибуты:

<code>max</code>	Максимальное значение прогресса задачи.
<code>min</code>	Минимальное значение прогресса задачи.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

class Status

Базовые классы: `Enum`

Состояние задачи.

Атрибуты:

<code>IDLE</code>	В ожидании.
<code>RUNNING</code>	Запущена.
<code>SUCCESS</code>	Выполнена успешно.
<code>CANCELED</code>	Отменена.
<code>ERROR</code>	Завершилась с ошибкой.

`__init__` (function: `Callable`, name: `str` = "", description: `str` = "", title: `str` = "", message: `str` = "", cancelable: `bool` = False, delayed: `bool` = True)

Конструктор класса.

Создает экземпляр класса.

Параметры

- **function** – Функция.
- **name** – Имя.
- **description** – Описание.
- **title** – Отображаемое имя.
- **message** – Отображаемое описание.
- **cancelable** – Добавить кнопку отмены в диалог.
- **delayed** – Добавить задержку перед показом диалога.

`__repr__()` → `str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`cancel()`

Уведомляет задачу об отмене.

Устанавливает свойство `axipy.Task.is_canceled` на `True` и меняет состояние задачи `axipy.Task.status` на `axipy.Task.Status.CANCELED`. Вызов метода при статусе `axipy.Task.Status.IDLE` также удалит задачу из `axipy.task_manager`. Метод является слотом, `Slot()` (`PySide2.QtCore.Slot`).

property cancelable: `bool`

Возвращает `True`, если в диалоге есть кнопка отмены.

property delayed: `bool`

Возвращает `True`, если в диалоге есть задержка перед показом.

property description: str

Возвращает описание задачи.

property finished: Signal

Возвращает сигнал, испускаемый при завершении задачи. Передает ссылку на экземпляр задачи, для получения результата, статуса и других атрибутов задачи.

Тип результата

Signal[axipy.Task]

property id: int

Возвращает идентификатор задачи.

Идентификатором задачи является её порядковый номер в менеджере задач `axipy.task_manager`.

property infinite_progress: bool

Возвращает или устанавливает бесконечную полосу прогресса для задачи.

Возвращает `True`, если бесконечный прогресс установлен, иначе возвращает `False`. При изменении значения на `False`, восстанавливает предыдущее значения прогресса. При изменениях значения свойства, сигнал `axipy.Task.range_changed` не вызывается.

property is_canceled: bool

Возвращает `True`, если задача была отменена.

Чтобы отменить задачу (назначить `True` для свойства `axipy.Task.is_canceled`), нужно вызвать `axipy.Task.cancel()`.

property max: float

Устанавливает или возвращает максимальное значение диапазона прогресса задачи.

property message: str

Устанавливает или возвращает отображаемое описание задачи.

property message_changed: Signal

Возвращает сигнал, испускаемый при изменении отображаемого описании задачи.

Тип результата

Signal[str]

property min: float

Устанавливает или возвращает минимальное значение диапазона прогресса задачи.

property name: str

Возвращает имя задачи.

raise_if_canceled(message: Optional[str] = None)

Вызывает исключение об отмене задачи `axipy.Task.CanceledException`.

Параметры

message – Сообщение, передаваемое в исключение.

property range: Range

Устанавливает или возвращает диапазон прогресса задачи.

property range_changed: Signal

Возвращает сигнал, испускаемый при изменении диапазона прогресса задачи.

Тип результата

Signal[axipy.Task.Range]

property result: Any

Возвращает результат задачи. (Возвращаемое значение пользовательской функции.) Если во время выполнения задачи, было вызвано исключение, то при получении результата, исключение будет вызвано повторно (За исключением пользовательской отмены `axipy.Task.CanceledException`).

Исключение

Exception – Если во время выполнения задачи, было вызвано исключение.

run_and_get(*args: Any, **kwargs: Any) → Any

Запускает задачу и ждет результат. Останавливается лишь выполнение кода python, цикл обработки событий Qt продолжает работу. (Сигналы и события продолжают обрабатываться.) Возвращает результат задачи. (Возвращаемое значение пользовательской функции.) Если во время выполнения задачи, было вызвано исключение, то при получении результата, исключение будет вызвано повторно (За исключением пользовательской отмены `axipy.Task.CanceledException`).

Параметры

- ***args** – Параметры, передаваемые в пользовательскую функцию.
- ****kwargs** – Именованные параметры, передаваемые в пользовательскую функцию.

Результат

Результат выполнения задачи. (Возвращаемое значение пользовательской функции.)

Исключение

Exception – Если во время выполнения задачи, было вызвано исключение.

start(*args: Any, **kwargs: Any)

Запускает задачу без ожидания результата. Чтобы получить результат, необходимо подписаться на сигнал `axipy.Task.finished` и обратиться к свойству `axipy.Task.result`.

Параметры

- ***args** – Параметры, передаваемые в пользовательскую функцию.
- ****kwargs** – Именованные параметры, передаваемые в пользовательскую функцию.

property started: Signal

Возвращает сигнал, испускаемый при старте задачи.

Тип результата

Signal[]

property status: Status

Возвращает текущее состояние задачи.

property thread_id: Optional[int]

Возвращает идентификатор потока, в котором выполняется задача.

Когда задача не выполняется, возвращает None.

property title: str

Устанавливает или возвращает отображаемое имя задачи.

property title_changed: Signal

Возвращает сигнал, испускаемый при изменении отображаемого имени задачи.

Тип результата

Signal[str]

property value: float

Устанавливает или возвращает текущее значение прогресса задачи.

property value_changed: Signal

Возвращает сигнал, испускаемый при изменении текущего значения прогресса задачи.

Тип результата

Signal[str]

20.3 TaskManager - Сервис для отслеживания пользовательских задач

class axipy.TaskManager

Менеджер пользовательских задач. Класс является словарем, доступным только для чтения (`collections.abc.Mapping`), где ключи это идентификаторы действий `axipy.Task.id`, а значения это объекты класса `axipy.Task`. Поддерживает обращение по ключу.

Примечание: Создание `axipy.TaskManager` не требуется, используйте объект `axipy.task_manager`.

Задачи (объекты класса `axipy.Task`) добавляются в менеджер при создании конструктором, и удаляются при завершении задачи (успешно или с ошибкой). Еще не запущенную задачу, можно удалить из менеджера, вызвав метод `axipy.Task.cancel`.

Методы:

`generate_dialog_for_task(task, spec)`**Предупреждение:**

Не
рекомендуется,
начиная
с
версии
6.0.0.

`get(key[, default_value])`
`items()`

Возвращает значение по ключу.

Возвращает набор кортежей
ключ-значение, где ключи это
идентификаторы задач `axipy.Task`
`.id`, а значения это объекты класса
`axipy.Task`.

`keys()`

Возвращает набор ключей, где ключи
это идентификаторы задач `axipy.Task`
`.id`.

`run_and_get(spec, func, *args, **kwargs)`**Предупреждение:**

Не
рекомендуется,
начиная
с
версии
6.0.0.

`run_in_gui(func, *args, **kwargs)`**Предупреждение:**

Не
рекомендуется,
начиная
с
версии
6.0.0.

`start_task(task)`**Предупреждение:**

Не
рекомендуется,
начиная
с
версии
6.0.0.

`values()`

Возвращает коллекцию значений, где
значения это объекты класса `axipy`
`.Task`.

Сигналы:

<code>added</code>	Возвращает сигнал, испускаемый при добавлении (создании) новой задачи.
<code>removed</code>	Возвращает сигнал, испускаемый при удалении (завершении) задачи.

Специальные методы:

<code>__getitem__(key)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__getitem__(key: int) → Task`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[int]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

property `added`: **Signal**

Возвращает сигнал, испускаемый при добавлении (создании) новой задачи.

Тип результата

`Signal[axipy.Task]`

`generate_dialog_for_task(task:`

`~axipy.concurrent.task._apply_deprecated.<locals>.AxipyTask,`

`spec:`

`~axipy.concurrent.task_utils.ProgressSpecification) →`

`QDialog`

Предупреждение: Не рекомендуется, начиная с версии 6.0.0.

`get(key: str, default_value: Optional[Any] = None) → Optional[Task]`

Возвращает значение по ключу.

`items() → ItemsView[int, Task]`

Возвращает набор кортежей ключ-значение, где ключи это идентификаторы задач `axipy.Task.id`, а значения это объекты класса `axipy.Task`.

`keys() → KeysView[int]`

Возвращает набор ключей, где ключи это идентификаторы задач `axipy.Task.id`.

property `removed`: **Signal**

Возвращает сигнал, испускаемый при удалении (завершении) задачи.

Тип результата

`Signal[axipy.Task]`

`run_and_get`(spec: ProgressSpecification, func: Callable, *args: Any, **kwargs: Any) → Any

Предупреждение: Не рекомендуется, начиная с версии 6.0.0.

`run_in_gui`(func: Callable, *args: Any, **kwargs: Any) → Any

Предупреждение: Не рекомендуется, начиная с версии 6.0.0.

`start_task`(task: ~axipy.concurrent.task._apply_deprecated.<locals>.AxipyTask)

Предупреждение: Не рекомендуется, начиная с версии 6.0.0.

`values`() → ValuesView[Task]

Возвращает коллекцию значений, где значения это объекты класса `axipy.Task`.

axipy.utl - Вспомогательные классы.

Вспомогательные классы.

21.1 Pnt - Точка

class axipy.Pnt

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).

Список 1: Создание точки.

```
from axipy import Pnt

# Создание точки
print(Pnt(1, 2))
print(Pnt(1.5, 2.5))
'''
>>> (1.0 2.0)
>>> (1.5 2.5)
'''
```

Конструктор класса:

<code>__init__(x, y)</code>	Конструктор класса.

Классовые методы:

<code>eq_approx(point1, point2[, precision])</code>	Сравнивает две точки с заданной точностью.
<code>from_qt(p)</code>	Преобразует из формата Qt.
<code>from_tuple(value)</code>	Создает точку из кортежа координат.

Свойства:

<code>x</code>	Устанавливает координату X.	или	возвращает
<code>y</code>	Устанавливает координату Y.	или	возвращает

Методы:

<code>clone()</code>	Возвращает копию.
<code>to_qt()</code>	Преобразование в формат Qt.
<code>to_tuple()</code>	Возвращает координаты точки как кортеж.

Специальные методы:

<code>__copy__()</code>	Поверхностная копия объекта.
<code>__deepcopy__([memo_dict])</code>	Глубокая копия объекта.
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__format__(format_spec)</code>	Форматированное строковое представление.
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__copy__() → Pnt`

Поверхностная копия объекта. (функция `copy.copy()`)

`__deepcopy__(memo_dict: Optional[dict] = None) → Pnt`

Глубокая копия объекта. (функция `copy.deepcopy()`)

`__eq__(other: Union[Pnt, Tuple[float, float], Any]) → bool`

Оператор равенства (`==`).

`__format__(format_spec: str) → str`

Форматированное строковое представление.

`__init__(x: float, y: float)`

Конструктор класса.

Параметры

- `x` - X координата.
- `y` - Y координата.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone() → Pnt`

Возвращает копию.

classmethod eq_approx(point1: [Pnt](#), point2: [Pnt](#), precision: [float](#) = 1e-12) → [bool](#)

Сравнивает две точки с заданной точностью.

Параметры

- **point1** – Первая точка сравнения.
- **point2** – Вторая точка сравнения.
- **precision** – Точность сравнения.

Результат

True если точки равны.

classmethod from_qt(p: [Union](#)[[QPoint](#), [QPointF](#)]) → [Pnt](#)

Преобразует из формата Qt.

Параметры

p – Преобразуемая точка.

Список 2: Пример.

```
from PySide2.QtCore import QPoint, QPointF

# Создание точки из формата Qt
qpoint = QPoint(1, 2)
print(Pnt.from_qt(qpoint))
qpointf = QPointF(1.5, 2.5)
print(Pnt.from_qt(qpointf))
'''
>>> (1.0 2.0)
>>> (1.5 2.5)
'''
```

classmethod from_tuple(value: [Tuple](#)[[float](#), [float](#)]) → [Pnt](#)

Создает точку из кортежа координат.

to_qt() → [QPointF](#)

Преобразование в формат Qt.

Список 3: Пример.

```
# Представление точки в формате Qt
print(Pnt(1, 2).to_qt())
'''
>>> PySide2.QtCore.QPointF(1.000000, 2.000000)
'''
```

to_tuple() → [Tuple](#)[[float](#), [float](#)]

Возвращает координаты точки как кортеж.

property x: [float](#)

Устанавливает или возвращает координату X.

property y: [float](#)

Устанавливает или возвращает координату Y.

21.2 Rect - Прямоугольник

class axipy.Rect

Прямоугольник, который не обладает геопривязкой. Используется для различного вида запросов.

Список 4: Создание прямоугольника.

```
from axipy import Rect

rect = Rect(0, 0, 10, 5)
print(rect)
'''
>>> (0.0 0.0) (10.0 5.0)
'''
```

Конструктор класса:

<code>__init__(xmin, ymin, xmax, ymax)</code>	Конструктор класса.

Классовые методы:

<code>eq_approx(rect1, rect2[, precision])</code>	Сравнивает два прямоугольника с заданной точностью.
<code>from_qt(r)</code>	Преобразует из формата Qt.

Свойства:

<code>center</code>	Устанавливает или возвращает центр прямоугольника.
<code>height</code>	Возвращает высоту прямоугольника.
<code>is_empty</code>	Возвращает признак, что один или оба размера равны нулю.
<code>is_valid</code>	Возвращает признак, является ли прямоугольник правильным.
<code>width</code>	Возвращает ширину прямоугольника.
<code>xmax</code>	Устанавливает или возвращает максимальное значение X.
<code>xmin</code>	Устанавливает или возвращает минимальное значение X.
<code>ymax</code>	Устанавливает или возвращает максимальное значение Y.
<code>ymin</code>	Устанавливает или возвращает минимальное значение Y.

Методы:

<code>clone()</code>	Возвращает копию.
<code>contains(other)</code>	Содержит ли полностью в своих границах переданный объект.
<code>expanded(dx, dy)</code>	Возвращает прямоугольник, увеличенный на заданные величины.
<code>intersected(other)</code>	Возвращает общий для обоих прямоугольник.
<code>intersects(other)</code>	Проверяет, имеются ли пересечения с другим прямоугольником.
<code>merge(other)</code>	Возвращает прямоугольник, занимаемый обоими прямоугольниками.
<code>normalize()</code>	Исправляет прямоугольник, если его ширина или высота отрицательны.
<code>to_qt()</code>	Преобразование в формат Qt.
<code>translated(dx, dy)</code>	Возвращает прямоугольник, смещенный на заданную величину.

Специальные методы:

<code>__copy__()</code>	Поверхностная копия объекта.
<code>__deepcopy__(memo_dict)</code>	Глубокая копия объекта.
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__format__(format_spec)</code>	Форматированное строковое представление.
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__copy__() → Rect`

Поверхностная копия объекта. (функция `copy.copy()`)

`__deepcopy__(memo_dict: Optional[dict] = None) → Rect`

Глубокая копия объекта. (функция `copy.deepcopy()`)

`__eq__(other: object) → bool`

Оператор равенства (==).

`__format__(format_spec: str) → str`

Форматированное строковое представление.

`__init__(xmin: float, ymin: float, xmax: float, ymax: float)`

Конструктор класса.

Параметры

- **xmin** – Минимальное значение X.
- **ymin** – Минимальное значение Y.
- **xmax** – Максимальное значение X.
- **ymax** – Максимальное значение Y.

`__repr__()` → `str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property center: `Pnt`

Устанавливает или возвращает центр прямоугольника.

clone() → `Rect`

Возвращает копию.

contains(other: `Union[Pnt, Tuple[float, float], QPointF, Rect, Tuple[float, float, float, float], QRectF, QRect]`) → `bool`

Содержит ли полностью в своих границах переданный объект.

Параметры

other – Переданный объект - точка или прямоугольник.

Список 5: Пример.

```
r = Rect(0, 0, 10, 10)
print("contains (0, 0)", r.contains((0, 0)))
print("contains (15, 10)", r.contains((15, 10)))
print("contains (0, 0) (5, 5)", r.contains(Rect(0, 0, 5, 5)))
print("contains (0, 0) (15, 15)", r.contains(Rect(0, 0, 15, 15)))
...

>>> contains (0, 0) True
>>> contains (15, 10) False
>>> contains (0, 0) (5, 5) True
>>> contains (0, 0) (15, 15) False
...
```

classmethod eq_approx(rect1: `Rect`, rect2: `Rect`, precision: `float` = 1e-12) → `bool`

Сравнивает два прямоугольника с заданной точностью.

Параметры

- **rect1** – Первый прямоугольник сравнения.
- **rect2** – Второй прямоугольник сравнения.
- **precision** – Точность сравнения.

Результат

True если прямоугольники равны.

expanded(dx: `float`, dy: `float`) → `Rect`

Возвращает прямоугольник, увеличенный на заданные величины. Увеличение размеров производится по отношению к центру, который не меняется в результате операции.

Параметры

- **dx** – расширение по X.
- **dy** – расширение по Y.

Список 6: Пример.

```
r = Rect(0, 0, 10, 10)
print(r.expanded(10, 10))
...
>>> (-5.0 -5.0) (15.0 15.0)
...
```

classmethod `from_qt(r: Union[QRect, QRectF]) → Rect`

Преобразует из формата Qt.

Параметры

r – Преобразуемый прямоугольник.

property `height: float`

Возвращает высоту прямоугольника.

intersects(other: Rect) → Rect

Возвращает общий для обоих прямоугольник.

Параметры

other – Прямоугольник, с которым производится операция.

Список 7: Пример.

```
r1 = Rect(0, 0, 5, 10)
r2 = Rect(0, 0, 10, 5)
print(r1.intersected(r2))
...
>>> (0.0 0.0) (5.0 5.0)
...
```

intersects(other: Rect) → bool

Проверяет, имеются ли пересечения с другим прямоугольником.

Параметры

other – Прямоугольник, с которым производится операция.

property `is_empty: bool`

Возвращает признак, что один или оба размера равны нулю.

property `is_valid: bool`

Возвращает признак, является ли прямоугольник правильным.

merge(other: Rect) → Rect

Возвращает прямоугольник, занимаемый обоими прямоугольниками.

Параметры

other – Прямоугольник, с которым производится операция.

Список 8: Пример.

```
r1 = Rect(0, 0, 5, 10)
r2 = Rect(0, 0, 10, 5)
print(r1.merge(r2))
...
>>> (0.0 0.0) (10.0 10.0)
...
```

normalize()

Исправляет прямоугольник, если его ширина или высота отрицательны.

Список 9: Пример.

```
from PySide2.QtCore import QRect
r = Rect.from_qt(QRect(0, 5, -10, 10))
print(r)
print(r.is_valid)
r.normalize()
print(r)
print(r.is_valid)
'''
>>> (0.0 5.0) (-10.0 15.0)
>>> False
>>> (-10.0 5.0) (0.0 15.0)
>>> True
'''
```

to_qt() → QRectF

Преобразование в формат Qt.

translated(dx: float, dy: float) → Rect

Возвращает прямоугольник, смещенный на заданную величину.

Параметры

- **dx** – смещение по X.
- **dy** – смещение по Y.

Список 10: Пример.

```
r = Rect(0, 0, 10, 10)
print(r.translated(10, -10))
'''
>>> (10.0 -10.0) (20.0 0.0)
'''
```

property width: float

Возвращает ширину прямоугольника.

property xmax: float

Устанавливает или возвращает максимальное значение X.

property xmin: float

Устанавливает или возвращает минимальное значение X.

property ymax: float

Устанавливает или возвращает максимальное значение Y.

property ymin: float

Устанавливает или возвращает минимальное значение Y.

21.3 FloatCoord - Координаты с плавающей точкой.

class axipy.FloatCoord

Класс представляет собой координату в формате числа с плавающей точкой (float).
Для угловой координаты используется класс `axipy.AngleCoord`.

Конструктор класса:

<code>__init__(value)</code>	Конструктор класса.

Свойства:

<code>value</code>	Возвращает числовое значение в формате числа с плавающей точкой (float).

Методы:

<code>as_float_round(precision)</code>	Округляет число до заданной точности.
<code>as_float_round_signific([digits])</code>	Округляет число с указанием количества значащих цифр.
<code>as_string(*[, precision, locale, ...])</code>	Возвращает число в виде строки.

Специальные методы:

<code>__bool__()</code>	Интерпретация объекта как логического значения.
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__ge__(other)</code>	Оператор 'Больше или равно' (>=).
<code>__gt__(other)</code>	Оператор 'Больше' (>).
<code>__le__(other)</code>	Оператор 'Меньше или равно' (<=).
<code>__lt__(other)</code>	Оператор 'Меньше' (<).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__bool__() → bool`

Интерпретация объекта как логического значения.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__ge__(other: object) → bool`

Оператор „Больше или равно“ (>=).

`__gt__(other: object) → bool`

Оператор „Больше“ (>).

__init__(value: Union[float, int, str])

Конструктор класса.

Создает координату из значения в различных форматах.

Параметры

value – Значение может быть:

- целым числом;
- числом с плавающей точкой;
- строкой, представляющей целое число или число с плавающей точкой;
- строкой, представляющей угловую координату с разделителями, или в формате румбов.

(<dd°mm'ss,zz">, <dd mm ss,zz>, <dd/mm/ss,zz>, <dd-mm-ss,zz>, <dd,mm,ss. zz>, <dd. zz>, <dd,zz> или в румбах ЮВ dd. zz.)

Исключение

ValueError – если не удалось преобразовать значение в число с плавающей точкой.

__le__(other: object) → bool

Оператор „Меньше или равно“ (<=).

__lt__(other: object) → bool

Оператор „Меньше“ (<).

__repr__() → str

Официальное строковое представление объекта при вызове `repr()`.

__str__() → str

Неформальное строковое представление объекта при вызове `str` или `print()`.

as_float_round(precision: int) → float

Округляет число до заданной точности.

Параметры

precision – Количество знаков после запятой.

as_float_round_signific(digits: int = 15) → float

Округляет число с указанием количества значащих цифр.

Параметры

digits – Количество значащих цифр.

as_string(*, precision: int = 15, locale: Optional[Union[QLocale, Language]] = None, omit_group_separator: bool = True, group_separator: Optional[str] = None, decimal_point: Optional[str] = None, suppress_trailing_zeros: bool = True) → str

Возвращает число в виде строки.

Параметры

- **precision** – Необходимое число знаков после запятой.
- **locale** – Локаль в которой нужно вывести значение. По умолчанию используется текущая локаль: `QLocale()`.
- **omit_group_separator** – Исключить разделитель разрядов.

- **group_separator** - Использовать другой разделитель разрядов.
- **decimal_point** - Использовать другой десятичный разделитель.
- **suppress_trailing_zeros** - Не показывать завершающие нули.

property value: **float**

Возвращает числовое значение в формате числа с плавающей точкой (**float**).

21.4 AngleCoord - Угловые координаты.

class axipy.AngleCoord

Класс представляет собой угловую координату. Для координаты в формате числа с плавающей точкой (**float**) используется класс **axipy.FloatCoord**.

Угловую координату можно создать используя конструктор класса.

Список 11: Создание угловой координаты конструктором класса.

```
from axipy import AngleCoord

# Создание угловой координаты конструктором класса.
angle_coord = AngleCoord('33°22'28,11972")
print(angle_coord)
'''
>>> 33°22'28,11972"
'''
```

Также, угловую координату можно создать из составляющих **from_parts()**.

Конструктор класса:

<code>__init__(value)</code>	Конструктор класса.

Классовые методы:

<code>from_parts(degrees[, minutes, seconds])</code>	Создает угловую координату из составляющих.

Свойства:

<code>degrees</code>	Возвращает градусы.
<code>minutes</code>	Возвращает минуты.
<code>seconds</code>	Возвращает секунды.
<code>value</code>	Возвращает числовое значение в формате числа с плавающей точкой (float).

Методы:

<code>as_rumb([precision, suppress_trailing_zeros])</code>	Получение строкового значения угловой координаты в формате румбов.
<code>as_string([delimiter, precision, ...])</code>	Получение строкового значения угловой координаты.
<code>to_normalized([polar])</code>	Возвращает угловую координату, нормализованную в диапазоне [0; 360) или [-180; 180].

Специальные методы:

<code>__bool__()</code>	Интерпретация объекта как логического значения.
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__ge__(other)</code>	Оператор 'Больше или равно' (>=).
<code>__gt__(other)</code>	Оператор 'Больше' (>).
<code>__le__(other)</code>	Оператор 'Меньше или равно' (<=).
<code>__lt__(other)</code>	Оператор 'Меньше' (<).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__bool__() → bool`

Интерпретация объекта как логического значения.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__ge__(other: object) → bool`

Оператор „Больше или равно“ (>=).

`__gt__(other: object) → bool`

Оператор „Больше“ (>).

`__init__(value: Union[float, int, str])`

Конструктор класса.

Создает координату из значения в различных форматах.

Параметры

value – Значение может быть:

- целым числом;
- числом с плавающей точкой;
- строкой, представляющей целое число или число с плавающей точкой;
- строкой, представляющей угловую координату с разделителями, или в формате румбов.

(<dd°mm' ss, zz">, <dd mm ss, zz>, <dd/mm/ss, zz>, <dd-mm-ss, zz>, <dd, mm, ss. zz>, <dd. zz>, <dd, zz> или в румбах ЮВ dd. zz.)

Исключение

ValueError – если не удалось преобразовать значение в число с плавающей точкой.

__le__(other: **object**) → **bool**

Оператор „Меньше или равно“ (<=).

__lt__(other: **object**) → **bool**

Оператор „Меньше“ (<).

__repr__() → **str**

Официальное строковое представление объекта при вызове **repr()**.

__str__() → **str**

Неформальное строковое представление объекта при вызове **str** или **print()**.

as_rumb(precision: **Optional**[**int**] = None, suppress_trailing_zeros: **bool** = False) → **str**

Получение строкового значения угловой координаты в формате румбов.

Параметры

- **precision** – Количество знаков после запятой. Если None, округление не производится.
- **suppress_trailing_zeros** – Признак удаления завершающих нулей.

as_string(delimiter: **Optional**[**str**] = None, precision: **Optional**[**int**] = None, suppress_trailing_zeros: **bool** = False) → **str**

Получение строкового значения угловой координаты.

Параметры

- **delimiter** – Разделитель. Например, '-' или '/'.
- **precision** – Количество знаков после запятой. Если None, округление не производится.
- **suppress_trailing_zeros** – Признак удаления завершающих нулей.

property degrees: int

Возвращает градусы.

classmethod from_parts(degrees: **int**, minutes: **int** = 0, seconds: **float** = 0.0) → **AngleCoord**

Создает угловую координату из составляющих.

Параметры

- **degrees** – Градусы.
- **minutes** – Минуты.
- **seconds** – Секунды.

property minutes: int

Возвращает минуты.

property seconds: float

Возвращает секунды.

to_normalized(polar: bool = False) → [AngleCoord](#)

Возвращает угловую координату, нормализованную в диапазоне [0; 360) или [-180; 180].

Параметры

polar – Если True, то для нормализации используется полярная система координат [0; 360), если False, то нормализация происходит в диапазоне [-180; 180].

property value: [float](#)

Возвращает числовое значение в формате числа с плавающей точкой ([float](#)).

21.5 CalcMode - Режим расчетов

class ахіру.CalcMode

Режим измерения расстояний и площадей.

Атрибуты:

PLANE	В плане.
SPHERE	На сфере.
ELLIPSOID	На эллипсоиде.

axipy.da - Модуль источников данных.

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

22.1 DataProvider - Провайдеры

22.1.1 ProviderManager - Объект открытия/создания данных

class axipy.ProviderManager

Класс открытия/создания объектов данных `DataProvider`.

Примечание: Создание `axipy.ProviderManager` не требуется, используйте объект `axipy.provider_manager`.

Примечание: Для удобного задания параметров используйте экземпляры провайдеров: `tab`, `shp`, `csv`, `mif`, `excel`, `sqlite`, `postgre`, `oracle`, `mssql`, `ogr`, `svg`, `gdal`, `rest`, `tms`, `wms`, `wmts`, `dwg`, `panorama`.

Примечание: Открытые данные автоматически попадают в хранилище данных `axipy.DataManager`.

Пример открытия локальной таблицы:

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

Свойства:

csv	Файловый провайдер — Текст с разделителями.
dwg	Провайдер данных AutoCAD.
excel	Провайдер чтения файлов Excel.
gdal	Растровый провайдер GDAL.
mif	Провайдер данных MIF-MID.
mssql	Провайдер для базы данных MSSQLServer.
ogr	Векторный провайдер OGR.
oracle	Провайдер для базы данных Oracle.
panorama	Провайдер данных ГИС Панорама.
postgre	Провайдер для базы данных PostgreSQL.
rest	Провайдер REST.
shp	Векторный провайдер SHP.
sqlite	Векторный провайдер sqlite.
svg	Провайдер для SVG.
tab	Провайдер MapInfo.
tms	Тайловый провайдер.
wms	Web Map Service.
wmts	Web Map Tile Service.

Методы:

<code>create(definition)</code>	Создает и открывает данные из описания.
<code>create_open(definition)</code>	Создает и открывает данные из описания.
<code>createfile(filepath, schema, *args, **kwargs)</code>	Создает таблицу.
<code>find_for_extension(file_extension)</code>	Возвращает список провайдеров, поддерживающих расширение.
<code>loaded_providers()</code>	Возвращает список всех загруженных провайдеров данных.
<code>open(definition)</code>	Открывает данные по описанию.
<code>open_hidden(definition)</code>	Открывает данные по описанию.
<code>openfile(filepath, *args, **kwargs)</code>	Открывает данные из файла.
<code>providers()</code>	Возвращает список всех загруженных провайдеров данных.
<code>read_contents(definition)</code>	Читает содержимое источника данных.

create(definition: dict) → [DataObject](#)

Создает и открывает данные из описания.

Параметры

definition – Описание объекта данных.

Псевдоним `create_open()`.

create_open(definition: dict) → [DataObject](#)

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.
- **hidden** - Если указано True, то созданный объект не будет зарегистрирован в каталоге. См. также `open_hidden()`
- **override** - Если указано True, то при наличии файла в файловой системе, он перезаписывается. В противном случае вызывается исключение.

Параметры

definition - Описание объекта данных.

Пример:

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

createfile(filepath: `str`, schema: `Schema`, *args: `Any`, **kwargs: `Any`) → `DataObject`

Создает таблицу.

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

Параметры

- **filepath** - Путь к создаваемой таблице.
- **schema** - Схема таблицы.

property csv: `CsvDataProvider`

Файловый провайдер — Текст с разделителями.

property dwg: `DwgDataProvider`

Провайдер данных AutoCAD.

property excel: `ExcelDataProvider`

Провайдер чтения файлов Excel.

find_for_extension(file_extension: `str`) → `List[DataProvider]`

Возвращает список провайдеров, поддерживающих расширение.

Параметры

file_extension - Расширение файла.

property gdal: `GdalDataProvider`

Растровый провайдер GDAL.

loaded_providers() → `Dict[str, str]`

Возвращает список всех загруженных провайдеров данных.

Результат

Провайдеры в виде пар (Идентификатор : Описание).

property mif: [MifMidDataProvider](#)

Провайдер данных MIF-MID.

property mssql: [MsSqlDataProvider](#)

Провайдер для базы данных MSSQLServer.

property ogr: [OgrDataProvider](#)

Векторный провайдер OGR.

open(definition: [dict](#)) → [DataObject](#)

Открывает данные по описанию.

Формат описания объектов данных (набор и тип параметров) индивидуален для каждого провайдера данных, однако многие элементы используются для всех провайдеров данных. В нижеприведенной таблице можно определить какие параметр можно указывать при открытии того или иного источника. Т.е. допустимые параметры для конкретного провайдера указаны в соответствующем методе open для этого провайдера.

Таблица 1: Доступные провайдеры данных и ссылки на дополнительные параметры:

Провайдер	Краткое описание	Ссылка
tab	Провайдер MapInfo	<code>axipy.TabDataProvider.open()</code>
csv	Текст с разделителями	<code>axipy.CsvDataProvider.open()</code>
ogr	Векторный провайдер OGR	<code>axipy.OgrDataProvider.open()</code>
excel	Провайдер чтения файлов Excel	<code>axipy.ExcelDataProvider.open()</code>
shp	Векторный провайдер SHP	<code>axipy.ShapeDataProvider.open()</code>
sqlite	Векторный провайдер sqlite	<code>axipy.SqliteDataProvider.open()</code>
svg	Провайдер для SVG	<code>axipy.SvgDataProvider.open()</code>
gdal	Растровый провайдер GDAL	<code>axipy.GdalDataProvider.open()</code>
postgre	Провайдер для базы данных PostgreSQL	<code>axipy.PostgreDataProvider.open()</code>
oracle	Провайдер для базы данных Oracle	<code>axipy.OracleDataProvider.open()</code>
mssql	Провайдер для базы данных MSSQLServer	<code>axipy.MsSqlDataProvider.open()</code>
rest	Провайдер REST	<code>axipy.RestDataProvider.open()</code>
tms	Тайловый провайдер	<code>axipy.TmsDataProvider.open()</code>
wms	Web Map Service	<code>axipy.WmsDataProvider.open()</code>
wmts	Web Map Tile Service	<code>axipy.WmtsDataProvider.open()</code>
dwg	Провайдер файлов AutoCAD	<code>axipy.DwgDataProvider.open()</code>
panorama	Провайдер файлов	<code>axipy.PanoramaDataProvider.open()</code>

Также существуют параметры, которые допустимы независимо от типа провайдера

Таблица 2: Доступные провайдеры данных и ссылки на дополнительные параметры:

Параметр	Краткое описание
provider	Используемый провайдер. Допустимые значения можно получить <code>loaded_providers()</code> . Если не задан, то система пытается его определить самостоятельно.
src	Ссылка на источник. Как правило, это имя файла. Для конкретного провайдера может дублироваться под другим именем.
dataobject	Если источник содержит несколько таблиц, то имя конкретного указывается через данный параметр
alias	Псевдоним для открываемого источника данных. В системе открытый объект будет доступен по этому имени

Параметры

definition – Описание объекта данных.

Пример открытия файла (аналогичен `openfile()`):

```
json = {'src': '../path/to/datadir/world.tab'}
table_world = provider_manager.open(json)
```

или, что тоже самое:

```
json = {'filepath': '../path/to/datadir/world.tab'}
table_world = provider_manager.open(json)
```

Пример открытия файла с несколькими таблицами:

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

Пример открытия таблицы базы данных:

```
definition = {"host": "localhost",
              "db": "sample",
              "user": "postgres",
              "password": "postgres",
              "dataobject": "public.world",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"provider": "PgDataProvider"}
table = provider_manager.open(definition)
```

open_hidden(definition: dict) → DataObject

Открывает данные по описанию. Аналогична функции `open()`, за исключением того, что когда данный объект добавляется в каталог, он не учитывается в общем списке и от него из этого каталога не приходят события.

Примечание: См. также `open()`

Параметры

definition – Описание объекта данных.

Пример:

```
table = axipy.provider_manager.open_hidden({'src': 'world.tab'})
print(len(axipy.data_manager), axipy.data_manager.exists(table))
axipy.data_manager.remove(table)
>>> 0 True
```

openfile(filepath: Union[str, Path], *args: Any, **kwargs: Any) → DataObject

Открывает данные из файла.

Параметры

- **filepath** – Путь к открываемому файлу.
- ****kwargs** – Именованные аргументы. Возможные варианты от провайдера. Подробнее см. `open()`

Пример:

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

property oracle: OracleDataProvider

Провайдер для базы данных Oracle.

property panorama: PanoramaDataProvider

Провайдер данных ГИС Панорама.

property postgre: PostgreDataProvider

Провайдер для базы данных PostgreSQL.

providers() → List[DataProvider]

Возвращает список всех загруженных провайдеров данных.

Результат

Список провайдеров.

read_contents(definition: dict) → List[str]

Читает содержимое источника данных.

Обычно используется для источников, способных содержать несколько объектов данных.

Параметры

definition – Описание источника данных.

Результат

Имена объектов данных.

Пример:

```
contents = axipy.provider_manager.read_contents('../path/to/datadir/example.  
↳gpkg')  
print(contents)  
>>> ['world', 'worldcap']  
  
world = provider_manager.openfile('../path/to/datadir/example.gpkg',  
↳dataobject='world')
```

Пример с СУБД:

```
src = provider_manager.postgre.get_source(host='ip_server',  
                                          db_name='test',  
                                          user='user',  
                                          password='pass')  
  
contents = provider_manager.read_contents(src)  
print(contents)
```

Пример с файлом AutoCAD:

```
src = axipy.provider_manager.dwg.get_source("sample.dwg")  
contents = provider_manager.read_contents(src)  
print(contents)
```

property rest: [RestDataProvider](#)

Провайдер REST.

property shp: [ShapeDataProvider](#)

Векторный провайдер SHP.

property sqlite: [SqliteDataProvider](#)

Векторный провайдер sqlite.

property svg: [SvgDataProvider](#)

Провайдер для SVG.

property tab: [TabDataProvider](#)

Провайдер MapInfo.

property tms: [TmsDataProvider](#)

Тайловый провайдер.

property wms: [WmsDataProvider](#)

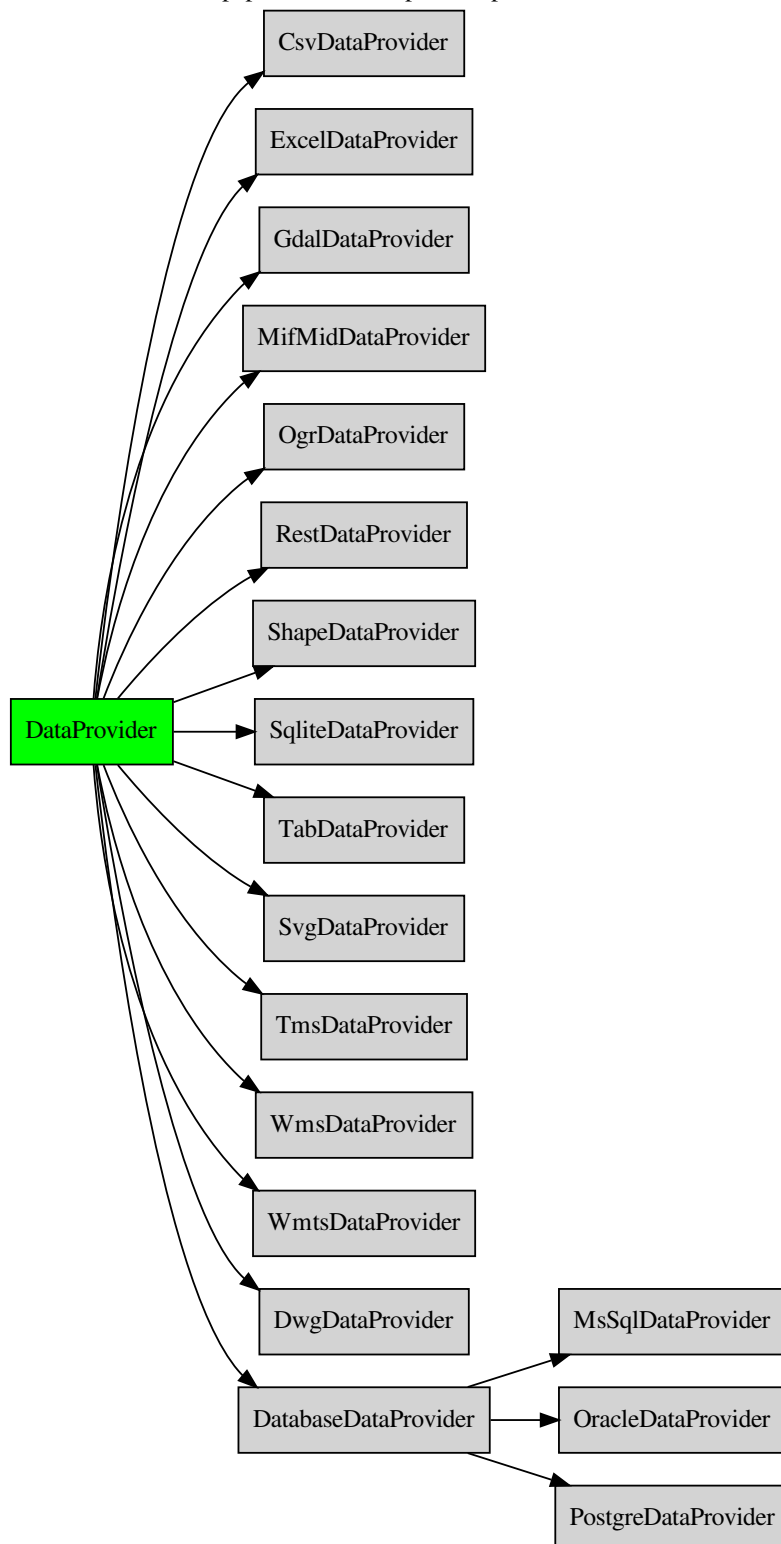
Web Map Service.

property wmts: [WmtsDataProvider](#)

Web Map Tile Service.

22.1.2 DataProvider - Провайдер данных

Иерархия классов провайдера данных:



class ахіру.DataProvider

Абстрактный провайдер данных.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open(*args, **kwargs)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(*args, **kwargs)</code>	Создает назначение объекта данных.
<code>get_source(*args, **kwargs)</code>	Создает источник данных.
<code>open(*args, **kwargs)</code>	Открывает объект данных.

create_open(*args: Any, **kwargs: Any) → DataObject

Создает и открывает объект данных.

Пример:

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

Что эквивалентно:

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

См.также:

DataProvider.destination().

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(*args: Any, **kwargs: Any) → Destination

Создает назначение объекта данных.

Исключение

NotImplementedError – Если провайдер не поддерживает создание назначений.

get_source(*args: Any, **kwargs: Any) → Source

Создает источник данных.

Исключение

NotImplementedError – Если провайдер не поддерживает создание источников.

property id: str

Идентификатор провайдера.

open(*args: Any, **kwargs: Any) → DataObject

Открывает объект данных.

Пример:

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

Что эквивалентно:

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

См.также:

`DataProvider.source()`.

22.1.3 Source - Источник данных

class `ахіру.Source`

Источник данных.

Используется для открытия данных или для указания источника при конвертации.

Список 1: Пример открытия и конвертации

```
file_in = 'world.tab'
file_out = 'world.mif'
source = provider_manager.tab.get_source(file_in).open()
destination = provider_manager.mif.get_destination(file_out, source.schema)
destination.export(source.items())
```

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

См.также:

[Импорт/Экспорт](#)

См.также:

[Destination](#)

Методы:

[open\(\)](#)

Открывает объект данных.

`open()` → [DataObject](#)

Открывает объект данных.

22.1.4 Destination - Назначение объекта данных

class `ахіру.Destination`

Назначение объекта данных.

Используется для создания данных или для указания назначения при конвертации.

Список 2: Пример открытия и конвертации

```
file_in = 'world.tab'
file_out = 'world.mif'
source = provider_manager.tab.get_source(file_in).open()
destination = provider_manager.mif.get_destination(file_out, source.schema)
destination.export(source.items())
```

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

См.также:

[Импорт/Экспорт](#)

См.также:

[Source](#)

Методы:

<code>create_open()</code>	Создает и открывает объект данных.
<code>export(features[, func_callback])</code>	Создает объект данных и экспортирует в него записи.
<code>export_from(source[, copy_schema])</code>	Создает объект данных и экспортирует в него записи из источника данных.
<code>export_from_table(table[, copy_schema, ...])</code>	Создает объект данных и экспортирует в него записи из таблицы.

`create_open()` → [DataObject](#)

Создает и открывает объект данных.

export(features: [Iterable](#)[[Feature](#)], func_callback: [Optional](#)[[Callable](#)[[[Feature](#), [int](#)], [Optional](#)[[bool](#)]]] = None)

Создает объект данных и экспортирует в него записи.

Параметры

- **features** – Записи.
- **func_callback** – Функция, которая будет вызываться после экспорта каждой записи. В определении должны быть параметры следующих типов:
 - feature [Feature](#) - текущая запись
 - row [int](#) - порядковый номер

Возможно прерывание процесса экспорта, для этого нужно вернуть False в func_callback.

Список 3: Пример экспорта данных

```
# Определяем схему будущей таблицы
schema = Schema(Attribute.string('name', 30), coordsystem="prj:1,104")
# Формируем данные для вставки. В нашем случае одна точка
```

(continues on next page)

(продолжение с предыдущей страницы)

```
features = [Feature(name='hello', geometry=Point(10, 10))]
# Имя выходного файла
filepath = './path/to/world_out.tab'
# Создаем таблицу по определенной ранее информации
dest = provider_manager.tab.get_destination(filepath, schema)
# Производим экспорт
dest.export(features)
```

export_from(source: [Source](#), copy_schema: [bool](#) = False)

Создает объект данных и экспортирует в него записи из источника данных.

Параметры

- **source** – Источник данных.
- **copy_schema** – Копировать схему источника без изменений.

export_from_table(table: [Table](#), copy_schema: [bool](#) = False, func_callback: [Optional](#)[[Callable](#)[[[Feature](#), [int](#)], [Optional](#)[[bool](#)]]] = None)

Создает объект данных и экспортирует в него записи из таблицы.

Параметры

- **table** – Таблица.
- **copy_schema** – Копировать схему источника без изменений.
- **func_callback** – Функция, которая будет вызываться после экспорта каждой записи. В определении должны быть параметры следующих типов:

- feature [Feature](#) - текущая запись
- row [int](#) - порядковый номер

Возможно прерывание процесса экспорта, для этого нужно вернуть False в func_callback.

Список 4: Пример экспорта таблицы с прогрессом

```
# Определяем функцию прогресса
def func(feature: Feature, row: int) -> Union[None, bool]:
    # Экспорт первых 10 записей, затем процесс прерывается
    if row == 10:
        return False # Прерывание процесса
    print(f'>> feature.id={feature.id} row={row}')

# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
destination = provider_manager.tab.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True, func_callback=func)
```

Список 5: Пример экспорта таблицы в формат CSV

```
# Открываем исходную таблицу
table_src = provider_manager.openfile(input_filepath)
# Формируем целевую и производим экспорт
```

(continues on next page)

(продолжение с предыдущей страницы)

```
destination = provider_manager.csv.get_destination(output_filepath, Schema())
destination.export_from_table(table_src, copy_schema=True)
```

22.1.5 ExportParameters - Дополнительные параметры экспорта

class ахіру.ExportParameters

Дополнительные параметры экспорта в таблицу базы данных.

См.также:

Пример использования см в главе [Импорт/Экспорт](#)

Атрибуты:

<code>createIndex</code>	Создавать пространственный индекс.
<code>dropTable</code>	Предварительно удалять существующую таблицу, если она присутствует в БД.
<code>errorFile</code>	Наименование файла, куда будут прописываться команды по вставке записей, не принятых сервером.
<code>fixGeometry</code>	Пробовать исправлять невалидную геометрию.
<code>geometryAsText</code>	Геометрию экспортировать как текстовые объекты.
<code>geometryColumnName</code>	Наименование геометрической колонки.
<code>logFile</code>	Наименование файла, куда будут прописываться успешно выполненные команды.
<code>mapCatalog</code>	Регистрация импортируемой таблицы в <code>mapinfo.mapinfo_mapcatalog</code> .
<code>renditonColumnName</code>	Наименование колонки с оформлением.
<code>srid</code>	Значение SRID.

`createIndex: bool`

Создавать пространственный индекс.

`dropTable: bool`

Предварительно удалять существующую таблицу, если она присутствует в БД.

`errorFile: Optional[str]`

Наименование файла, куда будут прописываться команды по вставке записей, не принятых сервером.

`fixGeometry: bool`

Пробовать исправлять невалидную геометрию.

`geometryAsText: bool`

Геометрию экспортировать как текстовые объекты.

geometryColumnName: `Optional[str]`

Наименование геометрической колонки.

logFile: `Optional[str]`

Наименование файла, куда будут прописываться успешно выполненные команды.

mapCatalog: `bool`

Регистрация импортируемой таблицы в `mapinfo.mapinfo_mapcatalog`.

renditonColumnName: `Optional[str]`

Наименование колонки с оформлением.

srid: `int`

Значение SRID.

22.1.6 CsvDataProvider - Текст с разделителями

class `axipy.CsvDataProvider`

Базовые классы: `DataProvider`

Файловый провайдер: Текст с разделителями.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.csv`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open(filepath, with_header, ...)</code>	<code>schema[, ...]</code>	Создает и открывает объект данных.
<code>file_extensions()</code>		Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema[, ...])</code>		Создает назначение объекта данных.
<code>get_source(filepath[, with_header, ...])</code>		Создает источник данных.
<code>open(filepath[, with_header, delimiter, ...])</code>		Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

create_open(filepath: `str`, schema: `Schema`, with_header: `bool` = True, delimiter: `str` = ';', encoding: `str` = 'utf8') → `Table`

Создает и открывает объект данных.

Параметры

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

file_extensions() → `List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: `str`, schema: `Schema`, with_header: `bool` = True, delimiter: `str` = ';', encoding: `str` = 'utf8') → `Destination`

Создает назначение объекта данных.

Параметры

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

get_source(filepath: `str`, with_header: `bool` = True, delimiter: `str` = ',', encoding: `str` = 'utf8', alias: `Optional[str]` = None) → `Source`

Создает источник данных.

Параметры

- **filepath** - Путь к файлу.
- **with_header** - Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** - Разделитель полей.
- **encoding** - Кодировка.
- **alias** - Псевдоним для открываемой таблицы.

property id: `str`

Идентификатор провайдера.

open(filepath: `str`, with_header: `bool` = True, delimiter: `str` = ',', encoding: `str` = 'utf8', alias: `Optional[str]` = None) → `Table`

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **delimiter** – Разделитель полей.
- **encoding** – Кодировка.
- **alias** – Псевдоним для открываемой таблицы.

22.1.7 ExcelDataProvider - Провайдер чтения файлов Excel

class axipy.ExcelDataProvider

Базовые классы: [DataProvider](#)

Провайдер чтения файлов Excel.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.excel`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open(filepath, schema)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema)</code>	Создает назначение объекта данных.
<code>get_source(filepath[, page, with_header, ...])</code>	Создает источник данных.
<code>open(filepath[, page, with_header, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

create_open(filepath: [str](#), schema: [Schema](#)) → [Table](#)

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

file_extensions() → [List\[str\]](#)

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: [str](#), schema: [Schema](#)) → [Destination](#)

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.

get_source(filepath: [str](#), page: [Optional\[str\]](#) = None, with_header: [bool](#) = False, encoding: [str](#) = 'utf8', alias: [Optional\[str\]](#) = None) → [Source](#)

Создает источник данных.

Параметры

- **filepath** – Путь к файлу.
- **page** – Имя страницы. Если не указана, то берется первая.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **encoding** – Кодировка.
- **alias** – Псевдоним для открываемой таблицы.

property id: [str](#)

Идентификатор провайдера.

open(filepath: [str](#), page: [Optional\[str\]](#) = None, with_header: [bool](#) = False, encoding: [str](#) = 'utf8', alias: [Optional\[str\]](#) = None) → [Table](#)

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **page** – Имя страницы.
- **with_header** – Признак того, что в первой строке содержатся имена атрибутов таблицы.
- **encoding** – Кодировка.
- **alias** – Псевдоним для открываемой таблицы.

22.1.8 MifMidDataProvider -

class axipy.MifMidDataProvider

Базовые классы: [DataProvider](#)

Провайдер данных MIF-MID.

Примечание: Поддерживает экспорт только в TAB. См. [convert_to_tab\(\)](#).

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.mif`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>convert_to_tab(mif_filepath, tab_filepath)</code>	Конвертирует из MIF в TAB.
<code>create_open()</code>	

Внимание:
Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema)</code>	Создает назначение объекта данных.
<code>get_source()</code>	

Внимание:
Не поддерживается.

<code>open()</code>	

Внимание:
Не поддерживается.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

convert_to_tab(mif_filepath: `Union[str, Path]`, tab_filepath: `Union[str, Path]`)
Конвертирует из MIF в TAB.

Список 6: Пример экспорта

```
# Исходный файл MIF
mif_filepath = './path/to/world.mif'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Целевой файл TAB
tab_filepath = './path/to/world_out.tab'
# Преобразование MIF в TAB
provider_manager.mif.convert_to_tab(mif_filepath, tab_filepath)
```

Параметры

- **mif_filepath** - Путь к исходному файлу.
- **tab_filepath** - Путь к выходному файлу.

Исключение

Exception - Если при конвертации произошла ошибка.

create_open() → **NoReturn**

Внимание: Не поддерживается.

Исключение

NotImplementedError -

file_extensions() → **List[str]**

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: Union[str, Path], schema: Schema) → **Destination**

Создает назначение объекта данных.

Параметры

- **filepath** - Путь к файлу.
- **schema** - Схема таблицы.

get_source() → **Source**

Внимание: Не поддерживается.

Исключение

NotImplementedError -

property id: str

Идентификатор провайдера.

open() → **NoReturn**

Внимание: Не поддерживается.

Исключение

NotImplementedError -

22.1.9 ShapeDataProvider - Векторный провайдер SHP

class axipy.ShapeDataProvider

Базовые классы: [DataProvider](#)

Векторный провайдер SHP.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.shp`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open(filepath, encoding]</code>	<code>schema[</code>	Создает и открывает объект данных.
<code>file_extensions()</code>		Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema[...])</code>		Создает назначение объекта данных.
<code>get_source(filepath[, encoding, prj, alias])</code>		Создает источник данных.
<code>open(filepath[, encoding, prj, alias])</code>		Открывает объект данных.
<code>open_temporary(schema)</code>		Создает и открывает временную таблицу.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

create_open(filepath: `str`, schema: [Schema](#), encoding: `str` = 'utf8') → [Table](#)

Создает и открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **schema** – Схема таблицы.
- **encoding** – Кодировка.

file_extensions() → [List\[str\]](#)

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: [str](#), schema: [Schema](#), encoding: [str](#) = 'utf8', create_data: [Optional\[dict\]](#) = None) → [Destination](#)

Создает назначение объекта данных.

Параметры

- **filepath** - Путь к файлу.
- **schema** - Схема таблицы.
- **encoding** - Кодировка.
- **create_data** - Параметры OGR. Разделены на два раздел, которые организуются как словари [dict](#).
 - »lco« - используется при создании слоя как дополнительные параметры. Соответствуют »Layer creation options«.
 - »dsc« - используется при создании источника данных как дополнительные параметры. Соответствуют »Database creation options«.

get_source(filepath: [str](#), encoding: [str](#) = 'utf8', prj: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None) → [Source](#)

Создает источник данных.

Параметры

- **filepath** - Путь к файлу.
- **encoding** - Кодировка.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.

property id: [str](#)

Идентификатор провайдера.

open(filepath: [str](#), encoding: [str](#) = 'utf8', prj: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None) → [Table](#)

Открывает объект данных.

Пример:

```
shp = provider_manager.shp.open('world.shp', prj='1, 104')
```

Параметры

- **filepath** - Путь к файлу.
- **encoding** - Кодировка.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.

open_temporary(schema: [Schema](#)) → [Table](#)

Создает и открывает временную таблицу.

Параметры

- **schema** - Схема таблицы.

22.1.10 SqliteDataProvider - Векторный провайдер sqlite

class axipy.SqliteDataProvider

Базовые классы: `DataProvider`

Векторный провайдер sqlite.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.sqlite`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open()</code>	
	Внимание: Не поддерживается.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, dataobject, schema)</code>	Создает назначение объекта данных.
<code>get_source(filepath[, dataobject, sql, prj, ...])</code>	Создает источник данных.
<code>open(filepath[, dataobject, sql, prj, alias])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: [str](#), dataobject: [str](#), schema: [Schema](#), geometry_format: [SqliteGeometryFormat](#) = [SqliteGeometryFormat.GPKG](#)) → [Destination](#)

Создает назначение объекта данных.

Args:

filepath: Источник данных или имя файла. dataobject: Наименование таблицы schema: Схема таблицы. geometry_format: Формат данных

См.также:

Пример использования см [Преобразование в формат Sqlite и GPKG](#)

get_source(filepath: [str](#), dataobject: [Optional\[str\]](#) = None, sql: [Optional\[str\]](#) = None, prj: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None) → [Source](#)

Создает источник данных. В качестве объекта может быть указана либо таблица, либо текст запроса. Если указан sql, то он имеет более высокий приоритет по отношению к значению dataobject. Если оба параметра опущены, будет возвращен None.

Параметры

- **filepath** - Путь к файлу.
- **dataobject** - Имя таблицы.
- **sql** - SQL-запрос.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.

Пример с таблицей:

```
table = provider_manager.openfile('world.sqlite', dataobject='world')
```

Пример с запросом и переопределенной СК:

```
table = provider_manager.openfile('world.sqlite', sql="select * from world_
↪where Страна like 'P%'", prj='12, 104, "m", 0')
```

property id: [str](#)

Идентификатор провайдера.

open(filepath: [str](#), dataobject: [Optional\[str\]](#) = None, sql: [Optional\[str\]](#) = None, prj: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None) → [Table](#)

Открывает объект данных.

В качестве объекта может быть указана либо таблица, либо текст запроса. Если указан sql, то он имеет более высокий приоритет по отношению к значению dataobject. Если оба параметра опущены, будет возвращен None.

Параметры

- **filepath** - Путь к файлу.
- **dataobject** - Имя таблицы.
- **sql** - SQL-запрос.

- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.

22.1.11 TabDataProvider - Провайдер MapInfo

class axipy.TabDataProvider

Базовые классы: [DataProvider](#)

Провайдер MapInfo.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.tab`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>change_coordsystem(filepath, coordsystem)</code>	Изменяет координатную систему в TAB файле без изменения самих данных.
<code>copy_table_files(src_filepath, dest_filepath)</code>	Копирует все связанные файлы с данным файлом в файловой системе под новым именем.
<code>create_open(filepath, schema)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema)</code>	Создает назначение объекта данных.
<code>get_source(filepath[, alias])</code>	Создает источник данных.
<code>open(filepath[, alias])</code>	Открывает объект данных.
<code>remove_table_files(filepath)</code>	Удаляет все связанные файлы с данным файлом в файловой системе.
<code>rename_table_files(src_filepath, dest_filepath)</code>	Переименовывает файл и все связанные файлы с ним.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

change_coordsystem(filepath: [Union\[str, Path\]](#), coordsystem: [CoordSystem](#))

Изменяет координатную систему в TAB файле без изменения самих данных. Меняется непосредственно сам файл, так что рекомендуется сделать копию.

Параметры

- **filepath** – Путь к файлу TAB (имя файла).

- **coordsystem** - Новое значение СК.

Исключение

RuntimeError - При возникновении ошибки

Список 7: Пример использования

```
in_filepath = 'path/to/input_filename.tab'
cs = CoordSystem.from_prj('10, 104, 7, 0')
provider_manager.tab.change_coordsystem(in_filepath, cs)
```

copy_table_files(src_filepath: Union[str, Path], dest_filepath: Union[str, Path])

Копирует все связанные файлы с данным файлом в файловой системе под новым именем.

Параметры

- **src_filepath** - Путь к исходному файлу TAB (имя файла).
- **dest_filepath** - Путь к выходному файлу TAB (имя файла).

Исключение

RuntimeError - При возникновении ошибки

Список 8: Пример использования

```
src_filepath = 'path/to/input_filename.tab'
dest_filepath = 'path/to/output_filename.tab'
provider_manager.tab.copy_table_files(src_filepath, dest_filepath)
```

create_open(filepath: str, schema: Schema) → Table

Создает и открывает объект данных.

Параметры

- **filepath** - Путь к файлу.
- **schema** - Схема таблицы.

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema) → Destination

Создает назначение объекта данных.

Параметры

- **filepath** - Путь к файлу.
- **schema** - Схема таблицы.

get_source(filepath: Union[str, Path], alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **filepath** - Путь к файлу.
- **alias** - Псевдоним для открываемой таблицы.

property id: `str`

Идентификатор провайдера.

open(filepath: `str`, alias: `Optional[str]` = None) → `Table`

Открывает объект данных.

Параметры

- **filepath** – Путь к файлу.
- **alias** – Псевдоним для открываемой таблицы.

remove_table_files(filepath: `Union[str, Path]`)

Удаляет все связанные файлы с данным файлом в файловой системе.

Параметры

filepath – Путь к файлу TAB (имя файла).

Исключение

`RuntimeError` – При возникновении ошибки

Список 9: Пример использования

```
filepath = 'path/to/input_filename.tab'
provider_manager.tab.remove_table_files(filepath)
```

rename_table_files(src_filepath: `Union[str, Path]`, dest_filepath: `Union[str, Path]`)

Переименовывает файл и все связанные файлы с ним.

Параметры

- **src_filepath** – Путь к исходному файлу TAB (имя файла).
- **dest_filepath** – Путь к новому имени файла TAB (имя файла). Файл не должен существовать.

Исключение

`RuntimeError` – При возникновении ошибки

Список 10: Пример использования

```
src_filepath = 'path/to/old_filename.tab'
dest_filepath = 'path/to/new_filename.tab'
provider_manager.tab.rename_table_files(src_filepath, dest_filepath)
```

22.1.12 SvgDataProvider - Провайдер для SVG

class `ахіру.SvgDataProvider`

Базовые классы: `DataProvider`

Провайдер для SVG.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.excel`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open()</code>	

Внимание:
Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.

<code>get_destination()</code>	

Внимание:
Не поддерживается.

<code>get_source(data[, alias])</code>	Создает источник данных.

<code>open(data[, alias])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

get_source(data: [str](#), alias: [Optional\[str\]](#) = None) → [Source](#)

Создает источник данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемой таблицы.

property id: [str](#)

Идентификатор провайдера.

open(data: [str](#), alias: [Optional\[str\]](#) = None) → [DataObject](#)

Открывает объект данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемой таблицы.

22.1.13 [PostgreDataProvider](#) - Провайдер для базы данных PostgreSQL

class [ахіру.PostgreDataProvider](#)

Базовые классы: [DatabaseDataProvider](#)

Провайдер для Базы Данных [PostgreSQL](#).

Пример с указанием имени таблицы:

```
definition = provider_manager.postgre.get_source('localhost', 'test', 'postgres',
↪ 'postgres', dataobject='world')
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.postgre.get_source('localhost', 'test', 'postgres',
↪ 'postgres', sql="select * from world where Страна like 'P%")
table = provider_manager.open(definition)
```

Примечание: Ссылку на провайдер можно получить через глобальную переменную [ахіру.provider_manager.postgre](#).

Свойства:

id	Идентификатор провайдера.

Атрибуты:

DEFAULT_PORT	Порт по умолчанию.

Методы:

<code>create_open(schema, dataobject, db_name, ...)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(schema, dataobject, db_name, ...)</code>	Создает назначение объекта данных.
<code>get_source(host, db_name, user, password[, ...])</code>	Создает описательную структуру для источника данных.
<code>open(host, db_name, user, password[, port, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

DEFAULT_PORT: `int`

Порт по умолчанию.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Table`

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Destination`

Создает назначение объекта данных.

Args:

schema: Схема таблицы. dataobject: Имя таблицы. db_name: Имя

базы данных. **host**: Адрес сервера. **user**: Имя пользователя. **password**: Пароль. **port**: Порт. **export_params**: Дополнительные параметры экспорта.

См.также:

Пример преобразования в БД

```
get_source(host: str, db_name: str, user: str, password: str, port: int =  
    DEFAULT_PORT, dataobject: Optional[str] = None, sql: Optional[str] =  
    None, prj: Optional[str] = None, alias: Optional[str] = None, unique:  
    Optional[str] = None, editable_unique: Optional[bool] = False) → Source
```

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных **ProviderManager.open()**.

В качестве таблицы можно указать либо ее наименование **dataobject** либо текст запроса **sql**.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению **dataobject**.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.
- **unique** – Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** – Установить возможность изменения значения поля **unique**.

property id: **str**

Идентификатор провайдера.

```
open(host: str, db_name: str, user: str, password: str, port: int = DEFAULT_PORT,  
    dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] =  
    None, alias: Optional[str] = None, unique: Optional[str] = None, editable_unique:  
    Optional[bool] = False) → Table
```

Открывает объект данных.

В качестве таблицы можно указать либо ее наименование **dataobject** либо текст запроса **sql**.

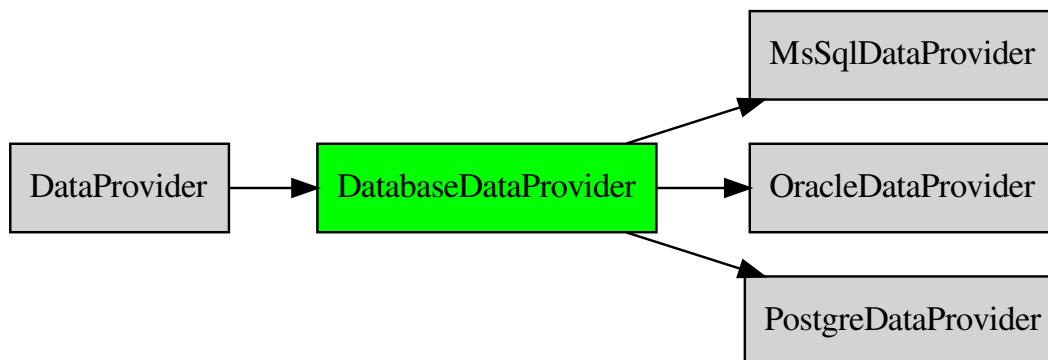
Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.

- **password** - Пароль.
- **port** - Порт.
- **dataobject** - Имя таблицы.
- **sql** - SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.
- **unique** - Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** - Установить возможность изменения значения поля unique.

22.1.14 DatabaseDataProvider - Провайдер для баз данных

Иерархия классов:



class axipy.DatabaseDataProvider

Базовые классы: DataProvider

Базовый класс для провайдеров БД.

Свойства:

<u>id</u>	Идентификатор провайдера.

Методы:

<code>create_open(schema, dataobject, db_name, ...)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(schema, dataobject, db_name, ...)</code>	Создает назначение объекта данных.
<code>get_source(host, db_name, user, password[, ...])</code>	Создает описательную структуру для источника данных.
<code>open(host, db_name, user, password[, port, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

create_open(schema: [Schema](#), dataobject: [str](#), db_name: [str](#), host: [str](#), user: [str](#), password: [str](#), port: [int](#) = DEFAULT_PORT, export_params: [Optional\[ExportParameters\]](#) = None) → [Table](#)

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

file_extensions() → [List\[str\]](#)

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(schema: [Schema](#), dataobject: [str](#), db_name: [str](#), host: [str](#), user: [str](#), password: [str](#), port: [int](#) = DEFAULT_PORT, export_params: [Optional\[ExportParameters\]](#) = None) → [Destination](#)

Создает назначение объекта данных.

Args:

schema: Схема таблицы. dataobject: Имя таблицы. db_name: Имя базы данных. host: Адрес сервера. user: Имя пользователя. password: Пароль. port: Порт. export_params: Дополнительные параметры экспорта.

См.также:

Пример преобразования в БД

```
get_source(host: str, db_name: str, user: str, password: str, port: int =
    DEFAULT_PORT, dataobject: Optional[str] = None, sql: Optional[str] =
    None, prj: Optional[str] = None, alias: Optional[str] = None, unique:
    Optional[str] = None, editable_unique: Optional[bool] = False) → Source
```

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных `ProviderManager.open()`.

В качестве таблицы можно указать либо ее наименование `dataobject` либо текст запроса `sql`.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению `dataobject`.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.
- **unique** – Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** – Установить возможность изменения значения поля `unique`.

property id: **str**

Идентификатор провайдера.

```
open(host: str, db_name: str, user: str, password: str, port: int = DEFAULT_PORT,
    dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] =
    None, alias: Optional[str] = None, unique: Optional[str] = None, editable_unique:
    Optional[bool] = False) → Table
```

Открывает объект данных.

В качестве таблицы можно указать либо ее наименование `dataobject` либо текст запроса `sql`.

Параметры

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

- **sql** - SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению `dataobject`.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.
- **unique** - Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** - Установить возможность изменения значения поля `unique`.

22.1.15 OracleDataProvider - Провайдер для базы данных Oracle

class `ахіру.OracleDataProvider`

Базовые классы: `DatabaseDataProvider`

Провайдер для Базы Данных Oracle.

Пример с указанием имени таблицы:

```
definition = provider_manager.oracle.get_source('localhost', 'test', 'oracle',  
↪ 'oracle', dataobject='world')  
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.oracle.get_source('localhost', 'test', 'oracle',  
↪ 'oracle', sql="select * from world where Страна like 'P%')  
table = provider_manager.open(definition)
```

Примечание: Для подключения к БД Oracle необходимо настроить Oracle Instant Client.

См. Руководство по установке и активации.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.oracle`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Атрибуты:

<code>DEFAULT_PORT</code>	Порт по умолчанию.

Методы:

<code>create_open(schema, dataobject, db_name, ...)</code>	Создает и открывает объект данных.
<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(schema, dataobject, db_name, ...)</code>	Создает назначение объекта данных.
<code>get_source(host, db_name, user, password[, ...])</code>	Создает описательную структуру для источника данных.
<code>open(host, db_name, user, password[, port, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

DEFAULT_PORT: `int`

Порт по умолчанию.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Table`

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Destination`

Создает назначение объекта данных.

Args:

schema: Схема таблицы. dataobject: Имя таблицы. db_name: Имя

базы данных. **host**: Адрес сервера. **user**: Имя пользователя. **password**: Пароль. **port**: Порт. **export_params**: Дополнительные параметры экспорта.

См.также:

Пример преобразования в БД

```
get_source(host: str, db_name: str, user: str, password: str, port: int =  
    DEFAULT_PORT, dataobject: Optional[str] = None, sql: Optional[str] =  
    None, prj: Optional[str] = None, alias: Optional[str] = None, unique:  
    Optional[str] = None, editable_unique: Optional[bool] = False) → Source
```

Создает описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных **ProviderManager.open()**.

В качестве таблицы можно указать либо ее наименование **dataobject** либо текст запроса **sql**.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению **dataobject**.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.
- **unique** – Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** – Установить возможность изменения значения поля **unique**.

property id: **str**

Идентификатор провайдера.

```
open(host: str, db_name: str, user: str, password: str, port: int = DEFAULT_PORT,  
    dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] =  
    None, alias: Optional[str] = None, unique: Optional[str] = None, editable_unique:  
    Optional[bool] = False) → Table
```

Открывает объект данных.

В качестве таблицы можно указать либо ее наименование **dataobject** либо текст запроса **sql**.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.

- **password** - Пароль.
- **port** - Порт.
- **dataobject** - Имя таблицы.
- **sql** - SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.
- **unique** - Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** - Установить возможность изменения значения поля unique.

22.1.16 MsSqlDataProvider - Провайдер для базы данных MSSQLServer

class ахіру.MsSqlDataProvider

Базовые классы: DatabaseDataProvider

Провайдер для Базы Данных MSSQLServer.

Пример с указанием имени таблицы:

```
definition = provider_manager.mssql.get_source('localhost', 'test', 'sa', 'sa',
↪dataobject='world')
table = provider_manager.open(definition)
```

Пример с указанием текста запроса:

```
definition = provider_manager.mssql.get_source('localhost', 'test', 'sa', 'sa',
↪sql="select * from world where Страна like 'P%')
table = provider_manager.open(definition)
```

Примечание: Для работы с СУБД Microsoft SQL Server необходимо скачать и установить Microsoft SQL Server Native Client.

См. Руководство по установке и активации.

Примечание: Ссылку на провайдер можно получить через глобальную переменную ахіру.provider_manager.mssql.

Свойства:

id	Идентификатор провайдера.

Атрибуты:

DEFAULT_PORT	Порт по умолчанию.

Методы:

<code>create_open(schema, db_name, ...)</code>	<code>dataobject</code>	Создает и открывает объект данных.
<code>file_extensions()</code>		Список поддерживаемых расширений файлов.
<code>get_destination(schema, db_name, ...)</code>	<code>dataobject</code>	Создает назначение объекта данных.
<code>get_source(host, db_name, user, password[, ...])</code>		Создает описательную структуру для источника данных.
<code>open(host, db_name, user, password[, port, ...])</code>		Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

DEFAULT_PORT: `int`

Порт по умолчанию.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Table`

Создает и открывает объект данных.

Параметры

- **schema** – Схема таблицы.
- **dataobject** – Имя таблицы.
- **db_name** – Имя базы данных.
- **host** – Адрес сервера.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **export_params** – Дополнительные параметры экспорта.

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination(schema: Schema, dataobject: str, db_name: str, host: str, user: str, password: str, port: int = DEFAULT_PORT, export_params: Optional[ExportParameters] = None) → Destination`

Создает назначение объекта данных.

Args:

schema: Схема таблиці. dataobject: Ім'я таблиці. db_name: Ім'я бази даних. host: Адрес сервера. user: Ім'я користувача. password: Пароль. port: Порт. export_params: Додаткові параметри експорту.

См.также:

Пример преобразования в БД

```
get_source(host: str, db_name: str, user: str, password: str, port: int =
    DEFAULT_PORT, dataobject: Optional[str] = None, sql: Optional[str] =
    None, prj: Optional[str] = None, alias: Optional[str] = None, unique:
    Optional[str] = None, editable_unique: Optional[bool] = False) → Source
```

Создаёт описательную структуру для источника данных. Она в дальнейшем может быть использована при открытии данных `ProviderManager.open()`.

В качестве таблицы можно указать либо её наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.
- **user** – Имя пользователя.
- **password** – Пароль.
- **port** – Порт.
- **dataobject** – Имя таблицы.
- **sql** – SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** – Строка Системы Координат.
- **alias** – Псевдоним для открываемой таблицы.
- **unique** – Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** – Установить возможность изменения значения поля unique.

property id: `str`

Идентификатор провайдера.

```
open(host: str, db_name: str, user: str, password: str, port: int = DEFAULT_PORT,
    dataobject: Optional[str] = None, sql: Optional[str] = None, prj: Optional[str] =
    None, alias: Optional[str] = None, unique: Optional[str] = None, editable_unique:
    Optional[bool] = False) → Table
```

Открывает объект данных.

В качестве таблицы можно указать либо её наименование dataobject либо текст запроса sql.

Параметры

- **host** – Адрес сервера.
- **db_name** – Имя базы данных.

- **user** - Имя пользователя.
- **password** - Пароль.
- **port** - Порт.
- **dataobject** - Имя таблицы.
- **sql** - SQL-запрос. Если указан, то он имеет более высокий приоритет по отношению к значению dataobject.
- **prj** - Строка Системы Координат.
- **alias** - Псевдоним для открываемой таблицы.
- **unique** - Поле с уникальным значением, которое будет использоваться как идентификатор.
- **editable_unique** - Установить возможность изменения значения поля unique.

22.1.17 TmsDataProvider - Тайловый провайдер

class ахіру.TmsDataProvider

Базовые классы: [DataProvider](#)

Провайдер для тайловых серверов.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.tms`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

`create_open()`

Внимание:
Не поддерживается.

`file_extensions()`

Список поддерживаемых расширений файлов.

`get_destination()`

Внимание:
Не поддерживается.

`get_source(templateUrl[, minLevel, ...])` Создает источник данных.

`open(templateUrl[, minLevel, maxLevel, ...])` Открывает объект данных.

Специальные методы:

`__repr__()`

Официальное строковое представление объекта при вызове `repr()`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

get_source(templateUrl: **str**, minLevel: **int** = 0, maxLevel: **int** = 19, size: **tuple** = (256, 256), type_address: **str** = 'xyz', watermark: **str** = '', watermark_style: **str** = '', prj: **Optional**[**str**] = None, live_time: **int** = 0, alias: **Optional**[**str**] = None, maxAttempts: **int** = 0, noLocalCache: **bool** = False, extra_data: **Optional**[**WebOpenData**] = None) → **Source**

Создает источник данных.

Параметры

- **templateUrl** - Шаблон для запроса данных. Например, `https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}.png`
- **minLevel** - Минимальный уровень показа
- **maxLevel** - Максимальный уровень показа
- **size** - Размер тайлов
- **type_address** - Тип адресации к тайлам. Поддерживается два значения: xyz и quadkey
- **watermark** - Ссылка на правообладателя
- **watermark_style** - Стиль оформления текста, с которым на карте будут отображаться данные о правообладателе.
- **prj** - Строка с Системой Координат. Если None, то используется значение по умолчанию (CoordSys Earth Projection 10, 157, „m“)
- **live_time** - время жизни тайла в секундах. Если равно 0, то значение не учитывается.
- **alias** - Псевдоним для открываемой таблицы.
- **maxAttempts** - Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.
- **noLocalCache** - Не использовать локальный кэш
- **extra_data** - Дополнительные параметры подключения

property id: **str**

Идентификатор провайдера.

open(templateUrl: **str**, minLevel: **int** = 0, maxLevel: **int** = 19, size: **tuple** = (256, 256), type_address: **str** = 'xyz', watermark: **str** = '', watermark_style: **str** = '', prj: **Optional**[**str**] = None, live_time: **int** = 0, alias: **Optional**[**str**] = None, maxAttempts: **int** = 0, noLocalCache: **bool** = False, extra_data: **Optional**[**WebOpenData**] = None) → **DataObject**

Открывает объект данных.

Параметры

- **templateUrl** - Шаблон для запроса данных. Например, `https://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}.png`
- **minLevel** - Минимальный уровень показа
- **maxLevel** - Максимальный уровень показа
- **size** - Размер тайлов
- **type_address** - Тип адресации к тайлам. Поддерживается два значения: xyz и quadkey

- **watermark** - Ссылка на правообладателя
- **watermark_style** - Стиль оформления текста, с которым на карте будут отображаться данные о правообладателе.
- **prj** - Строка с Системой Координат. Если None, то используется значение по умолчанию (CoordSys Earth Projection 10, 157, „m“)
- **live_time** - время жизни тайла в секундах. Если равно 0, то значение не учитывается.
- **alias** - Псевдоним для открываемого источника данных.
- **maxAttempts** - Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.
- **noLocalCache** - Не использовать локальный кэш
- **extra_data** - Дополнительные параметры подключения

Пример открытия источника:

```
prj_mercator = 'CoordSys Earth Projection 10, 104, "m", 0 Bounds (-20037508.
↪34, -20037508.34) (20037508.34, 20037508.34)'
osm_raster = provider_manager.tms.open('http://maps.axioma-gis.ru/osm/{LEVEL}/
↪{ROW}/{COL}.png', prj=prj_mercator)
osm_layer = Layer.create(osm_raster)
map = Map([ osm_layer ])
view_manager.create_mapview(map)
```

22.1.18 RestDataProvider - Провайдер REST

class axipy.RestDataProvider

Базовые классы: [DataProvider](#)

Провайдер для ArcGIS REST.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.rest`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

`create_open()`

Внимание:
Не поддерживается.

`file_extensions()`

Список поддерживаемых расширений файлов.

`get_destination()`

Внимание:
Не поддерживается.

`get_source(url[, fmt, imageSR, size, dpi, ...])` Создает источник данных.

`open(url[, fmt, imageSR, size, dpi, ...])` Открывает объект данных.

Специальные методы:

`__repr__()`

Официальное строковое представление объекта при вызове `repr()`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

get_source(url: `str`, fmt: `str` = 'png32', imageSR: `int` = 102100, size: `Union[str, tuple]` = (1024, 1024), dpi: `int` = 96, transparent: `str` = 'true', layers: `str` = '', alias: `Optional[str]` = None, maxAttempts: `int` = 0) → `Source`

Создает источник данных.

Параметры

- **url** - Базовый URL.
- **fmt** - Формат выходного растра.
- **imageSR** - Код EPSG для выходного растра.
- **size** - Размер тайлов.
- **dpi** - DPI.
- **transparent** - Прозрачность выходного растра.
- **layers** - Перечень слоев.
- **alias** - Псевдоним для открываемой таблицы.
- **maxAttempts** - Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.

property id: `str`

Идентификатор провайдера.

open(url: `str`, fmt: `str` = 'png32', imageSR: `int` = 102100, size: `Union[str, tuple]` = (1024, 1024), dpi: `int` = 96, transparent: `str` = 'true', layers: `str` = '', alias: `Optional[str]` = None, maxAttempts: `int` = 0) → `DataObject`

Открывает объект данных.

Параметры

- **url** - Базовый URL.
- **fmt** - Формат выходного растра.
- **imageSR** - Код EPSG для выходного растра.
- **size** - Размер тайлов.
- **dpi** - DPI.
- **transparent** - Прозрачность выходного растра.
- **layers** - Перечень слоев.
- **alias** - Псевдоним для открываемой таблицы.
- **maxAttempts** - Максимальное количество попыток запроса. Значение 0 соответствует значению по умолчанию.

22.1.19 WmsDataProvider - Web Map Service

class axipy.WmsDataProvider

Базовые классы: [DataProvider](#)

Провайдер для Web Map Service.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.wms`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>create_open()</code>	

Внимание:
Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.

<code>get_destination()</code>	

Внимание:
Не поддерживается.

<code>get_source(url_capabilities, layers[, ...])</code>	Создает источник данных.

<code>open(url_capabilities, layers[, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

file_extensions() → `List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination() → `NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

get_source(url_capabilities: `str`, layers: `List[str]`, image_format: `str` = 'image/png', prj: `Optional[str]` = None, style: `Optional[str]` = None, alias: `Optional[str]` = None, extra_data: `Optional[WebOpenData]` = None, color: `Optional[QColor]` = None, size: `Optional[Union[str, tuple]]` = None) → `Source`

Создает источник данных.

Параметры

- **url_capabilities** - URL с метаданными capabilities.
- **layers** - Перечень слоев в виде списка.
- **prj** - Строка Системы Координат
- **image_format** - Формат выходного растра.
- **style** - Наименование стиля оформления.
- **alias** - Псевдоним для открываемого источника данных.
- **extra_data** - Дополнительные параметры подключения
- **color** - Цвет подложки растра. Если None, устанавливается прозрачной
- **size** - Максимальный размер выходного растра.

property id: `str`

Идентификатор провайдера.

open(url_capabilities: `str`, layers: `List[str]`, image_format: `str` = 'image/png', prj: `Optional[str]` = None, style: `Optional[str]` = None, alias: `Optional[str]` = None, extra_data: `Optional[WebOpenData]` = None, color: `Optional[QColor]` = None, size: `Optional[Union[str, tuple]]` = None) → `DataObject`

Открывает объект данных.

Параметры

- **url_capabilities** - URL с метаданными capabilities.
- **layers** - Перечень слоев в виде списка.
- **prj** - Строка Системы Координат
- **image_format** - Формат выходного растра.
- **style** - Наименование стиля оформления.

- **alias** - Псевдоним для открываемого источника данных.
- **extra_data** - Дополнительные параметры подключения
- **color** - Цвет подложки растра. Если None, устанавливается прозрачной
- **size** - Максимальный размер выходного растра.

Пример:

```
wms_raster = provider_manager.wms.open('http://www.mapinfo.com/miwms', ['World', ''], prj='EPSG:4326', style='AreaStyleGreen')
```

22.1.20 WebOpenData - Дополнительные данные открытия WEB

class axipy.WebOpenData

Дополнительные данные, которые используются при открытии WEB источников.

Пример открытия источника WMS:

```
open_data = WebOpenData()
open_data.header = {"referer": "referer_value"}
open_data.authenticator = {"user": "nyuser", "password": "mypass"}
wms_raster = provider_manager.wms.open('http://www.mapinfo.com/miwms', ['World'], prj='EPSG:4326', style='AreaStyleGreen', extra_data=open_data)
```

Свойства:

authenticator	Устанавливает или возвращает словарь, содержащий данные для аутентификации.
header	Устанавливает или возвращает словарь, содержащий заголовки HTTP-запросов.

22.1.21 WmtsDataProvider - Web Map Tile Service

class axipy.WmtsDataProvider

Базовые классы: [DataProvider](#)

Провайдер для тайловых серверов.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.wmts`.

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

`create_open()`

Внимание:
Не поддерживается.

`file_extensions()`

Список поддерживаемых расширений файлов.

`get_destination()`

Внимание:
Не поддерживается.

`get_source(capabilitiesUrl, dataObject[, alias])` Создает источник данных.

`open(capabilitiesUrl, dataObject[, alias])` Открывает объект данных.

Специальные методы:

`__repr__()`

Официальное строковое представление объекта при вызове `repr()`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

get_source(capabilitiesUrl: [str](#), dataObject: [str](#), alias: [Optional\[str\]](#) = None) → [Source](#)

Создает источник данных.

Параметры

- **capabilitiesUrl** – URL запроса метаданных.
- **dataObject** – Наименование слоя.
- **alias** – Псевдоним для открываемой таблицы.

property id: [str](#)

Идентификатор провайдера.

open(capabilitiesUrl: [str](#), dataObject: [str](#), alias: [Optional\[str\]](#) = None) → [DataObject](#)

Открывает объект данных.

Параметры

- **capabilitiesUrl** – URL запроса метаданных.
- **dataObject** – Наименование слоя.
- **alias** – Псевдоним для открываемого источника данных.

22.1.22 GdalDataProvider - Растровый провайдер GDAL

class [axipy.GdalDataProvider](#)

Базовые классы: [DataProvider](#)

Провайдер для растров.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.gdal`.

Свойства:

id	Идентификатор провайдера.

Методы:

`create_open()`

Внимание:
Не поддерживается.

`file_extensions()`

Список поддерживаемых расширений файлов.

`get_destination()`

Внимание:
Не поддерживается.

`get_source(data[, alias, open_data])`

Создает источник данных.

`open(data[, alias, open_data])`

Открывает объект данных.

Специальные методы:

`__repr__()`

Официальное строковое представление объекта при вызове `repr()`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination() → NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

`get_source(data: str, alias: Optional[str] = None, open_data: Optional[dict] = None)`

`→ Source`

Создает источник данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемой таблицы.
- **open_data** – Параметры GDAL. Передаются как пары параметр-значения, которые в конечном итоге рассматриваются как дополнительные параметры при открытии в функции [GDALOpenEx](#). Конкретные параметры необходимо смотреть в разделе «Dataset open options» для рассматриваемого источника.

property id: `str`

Идентификатор провайдера.

open(data: `str`, alias: `Optional[str]` = None, open_data: `Optional[dict]` = None) → `Table`
Открывает объект данных.

Параметры

- **data** – Имя файла или описание источника данных.
- **alias** – Псевдоним для открываемого раstra.
- **open_data** – Параметры GDAL. Передаются как пары параметр-значения, которые в конечном итоге рассматриваются как дополнительные параметры при открытии в функции [GDALOpenEx](#). Конкретные параметры необходимо смотреть в разделе «Dataset open options» для рассматриваемого источника.

Пример открытия раstra:

```
raster = axipy.provider_manager.gdal.open("PG:host=server_name dbname='db_name'
↪ ' user='user_name' schema='public' table='truemarble'", open_data = {
↪ 'PASSWORD': 'password'})
```

22.1.23 OgrDataProvider - Векторный провайдер OGR

class `axipy.OgrDataProvider`

Базовые классы: `DataProvider`

Провайдер для векторных данных OGR.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `axipy.provider_manager.ogr`.

См. также:

Пример использования см [Преобразование с использованием библиотеки OGR](#)

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

`create_open()`

Внимание:

Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, dataobject, schema)</code>	Создает назначение объекта данных.
<code>get_source(data[, dataobject, alias, ...])</code>	Создает источник данных.
<code>open(data[, dataobject, alias, encoding, ...])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`create_open() → DataObject`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` -

`file_extensions() → List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

`get_destination(filepath: str, dataobject: str, schema: Schema, open_mode: OpenMode = OpenMode.Create, create_data: Optional[dict] = None) → Destination`

Создает назначение объекта данных.

Параметры

- **filepath** - Источник данных или имя файла.
- **dataobject** - Наименование таблицы
- **schema** - Схема таблицы.
- **open_mode** - Режим открытия файла. В случае `OpenMode.Append` будет производиться дополнение к существующему файлу.
- **create_data** - Параметры OGR. Разделены на два раздел, которые организуются так же как словари `dict`.

- »lco» - используется при создании слоя как дополнительные параметры. Соответствуют «Layer creation options» у рассматриваемого источника
- »dco» - используется при создании источника данных как дополнительные параметры. Соответствуют «Database creation options»
- »driver» - явное указание используемого драйвера»

См.также:

Преобразование с использованием библиотеки OGR

get_source(data: [str](#), dataobject: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None, encoding: [str](#) = 'utf8', open_data: [Optional\[dict\]](#) = None) → [Source](#)

Создает источник данных.

Параметры

- **data** - Источник данных или имя файла.
- **dataobject** - Наименование таблицы. Если не указано, будет выбрано по умолчанию.
- **alias** - Псевдоним для открываемой таблицы.
- **encoding** - Кодировка.
- **open_data** - Параметры OGR. Передаются как пары параметр-значения, которые в конечном итоге рассматриваются как дополнительные параметры при открытии в функции [GDALOpenEx](#) Конкретные параметры необходимо смотреть в разделе «Dataset open options» для рассматриваемого источника.

property id: [str](#)

Идентификатор провайдера.

open(data: [str](#), dataobject: [Optional\[str\]](#) = None, alias: [Optional\[str\]](#) = None, encoding: [str](#) = 'utf8', open_data: [Optional\[dict\]](#) = None) → [DataObject](#)

Открывает объект данных.

Параметры

- **data** - Источник данных или имя файла.
- **dataobject** - Наименование таблицы. Если не указано, будет выбрано по умолчанию.
- **alias** - Псевдоним для открываемой таблицы.
- **encoding** - Кодировка.
- **open_data** - Параметры OGR. Передаются как пары параметр-значения, которые в конечном итоге рассматриваются как дополнительные параметры при открытии в функции [GDALOpenEx](#) Конкретные параметры необходимо смотреть в разделе «Dataset open options» для рассматриваемого источника.

22.1.24 DwgDataProvider - Провайдер для AutoCAD

class ахіру.DwgDataProvider

Базовые классы: [DataProvider](#)

Провайдер для источников формата AutoCAD.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.dwg`.

[Пример преобразования из DWG и Панорамы](#)

[Пример преобразования в DWG и Панораму](#)

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>convert_file(src_filepath, dest_filepath[, ...])</code>	Производит конвертацию исходный файл текущего провайдера в другой формат этого же провайдера.
<code>create_open()</code>	

Внимание:

Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, schema[, ...])</code>	Создает назначение объекта данных.
<code>get_source(filename[, alias])</code>	Создает источник данных.
<code>open(filename[, alias])</code>	Открывает объект данных.
<code>set_palette(palette)</code>	Устанавливает текущую палитру.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

convert_file(src_filepath: `str`, dest_filepath: `str`, out_version: `DwgFileVersion` = `DwgFileVersion.AutoCAD_2010`, out_format: `DwgFileFormat` = `DwgFileFormat.Auto`)

Производит конвертацию исходный файл текущего провайдера в другой формат этого же провайдера.

Параметры

- **src_filepath** - Путь к исходному файлу (имя файла).
- **dest_filepath** - Путь к выходному файлу (имя файла).
- **out_version** - Версия выходного файла.
- **out_format** - Формат выходного файла.

```
input_file = 'filename_in.dwg'
output_file = 'filename_out.dxf'
provider_manager.dwg.convert_file(input_file, output_file, out_format =
↳ DwgFileFormat.Dxf,
    out_version = DwgFileVersion.AutoCAD_2010)
```

create_open() → NoReturn

Внимание: Не поддерживается.

Исключение

NotImplementedError -

file_extensions() → List[str]

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: str, schema: Schema, layer_name: Optional[str] = None, version: DwgFileVersion = DwgFileVersion.AutoCAD_2013, format: DwgFileFormat = DwgFileFormat.Auto, coordsystem: Optional[CoordSystem] = None, open_mode: OpenMode = OpenMode.Create, attributes: bool = True) → Destination

Создает назначение объекта данных.

Параметры

- **filepath** - Путь к результирующему файлу.
- **schema** - Схема таблицы.
- **layer_name** - Наименование слоя, в который будет экспортироваться данные. Если None, данные будут добавлены в слой „0“
- **version** - Версия выходного файла
- **format** - Формат выходного файла
- **coordsystem** - Система координат, в которой необходимо получить результат. Если не указана, берется из схемы.
- **open_mode** - Режим открытия файла. В случае OpenMode.Append будет производиться дополнение к существующему файлу.
- **attributes** - Экспортировать атрибуты.

get_source(filename: str, alias: Optional[str] = None) → Source

Создает источник данных.

Параметры

- **filename** - Имя файла.
- **alias** - Псевдоним для открываемой таблицы.

property id: **str**

Идентификатор провайдера.

open(filename: **str**, alias: **Optional[str]** = None) → **Table**

Открывает объект данных.

Параметры

- **filename** - Имя файла для открытия.
- **alias** - Псевдоним для открываемого объекта.

set_palette(palette: **DwgPalette**)

Устанавливает текущую палитру.

Параметры

palette - Индекс палитры.

22.1.25 DwgFileVersion - Версия файла AutoCAD

class axipy.**DwgFileVersion**

Версия файла DWG AutoCAD.

Используется при задании параметра в функции **DwgDataProvider.convert_file()**

Атрибуты:

AutoCAD_R9	AutoCAD Release 9.
AutoCAD_R10	AutoCAD Release 10.
AutoCAD_R11_12	AutoCAD Release 11-12.
AutoCAD_R13	AutoCAD Release 13.
AutoCAD_R14	AutoCAD Release 14.
AutoCAD_2000	AutoCAD 2000-2002.
AutoCAD_2004	AutoCAD 2004-2006.
AutoCAD_2007	AutoCAD 2007-2009.
AutoCAD_2010	AutoCAD 2010-2012.
AutoCAD_2013	AutoCAD 2013-2016.

22.1.26 DwgFileFormat - Формат файла AutoCAD

class axipy.**DwgFileFormat**

Формат файла AutoCAD.

Используется при задании параметра в функции **DwgDataProvider.convert_file()**

Атрибуты:

Dwg	Файл DWG.
Dxf	Текстовый файл DXF.
Dxb	Бинарный файл DXF.
Auto	Явно не задано.

22.1.27 DwgPalette - Палитра данных AutoCAD

class ахіру.DwgPalette

Палитра при работе с файлами AutoCAD.

Используется при задании параметра в функции `DwgDataProvider.set_palette()`

Атрибуты:

Light	Светлая тема.
Dark	Темная тема.

22.1.28 PanoramaDataProvider - Провайдер для ГИС Панорама

class ахіру.PanoramaDataProvider

Базовые классы: `DataProvider`

Провайдер для источников ГИС Панорама.

Примечание: Ссылку на провайдер можно получить через глобальную переменную `ахіру.provider_manager.panorama`.

Пример преобразования из DWG и Панорамы

Пример преобразования в DWG и Панораму

Свойства:

<code>id</code>	Идентификатор провайдера.

Методы:

<code>convert_file(src_filepath, dest_filepath, ...)</code>	Производит конвертацию исходного файла в формате SXF в другой формат этого же провайдера (MAP).
<code>create_open()</code>	

Внимание:

Не поддерживается.

<code>file_extensions()</code>	Список поддерживаемых расширений файлов.
<code>get_destination(filepath, classificator, schema)</code>	Создает назначение объекта данных.
<code>get_source(filename[, alias, empty_layers])</code>	Создает источник данных.
<code>is_supported_coordsystem(coordsystem)</code>	Производится проверка, поддерживается ли СК провайдером.
<code>open(filename[, alias, empty_layers])</code>	Открывает объект данных.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

convert_file(src_filepath: `Union[str, Path]`, dest_filepath: `Union[str, Path]`, classifier: `Union[str, Path]`)

Производит конвертацию исходного файла в формате SXF в другой формат этого же провайдера (MAP).

Параметры

- **src_filepath** – Путь к исходному файлу SXF (имя файла).
- **dest_filepath** – Путь к выходному файлу (имя файла).
- **classifier** – Путь к классификатору

```
input_file = 'Podolsk.sxf'
output_file = 'Podolsk.map'
classifier = 'Topol00t.rsc'
provider_manager.panorama.convert_file(input_file, output_file, classifier)
```

create_open() → `NoReturn`

Внимание: Не поддерживается.

Исключение

`NotImplementedError` –

file_extensions() → `List[str]`

Список поддерживаемых расширений файлов.

Результат

Пустой список для не файловых провайдеров.

get_destination(filepath: `Union[str, Path]`, classifier: `str`, schema: `Schema`, key_field: `Optional[str]` = `None`, single_object_type: `Optional[str]` = `None`, coordsystem: `Optional[CoordSystem]` = `None`, open_mode: `OpenMode` = `OpenMode.Create`) → `Destination`

Создает назначение объекта данных.

Параметры

- **filepath** – Путь к результирующему файлу SIT или MAP.
- **schema** – Схема таблицы.
- **classifier** – Путь к классификатору. При проведении конвертации копия классификатора так же сохраняется в папке с выходным файлом.
- **key_field** – Колонка-атрибут, содержащая ключи объектов, для файла с разнотипными объектами

- **single_object_type** - Символьный ключ объекта для файла, содержащего однотипные объекты
- **coordsystem** - Система координат, в которой необходимо получить результат. Если не указана, берется из схемы. Если СК не может быть преобразована в СК Панорамы, то вызывается исключение. Проверить предварительно можно воспользовавшись функцией `is_supported_coordsystem()`
- **open_mode** - Режим открытия файла. В случае `OpenMode.Append` будет производится дополнение к существующему файлу.

Примечание: Обязательно одно из двух полей `key_field` или `single_object_type`

get_source(filename: `Union[str, Path]`, alias: `Optional[str]` = None, empty_layers: `bool` = False) → `Source`

Создает источник данных.

Параметры

- **filename** - Имя файла.
- **alias** - Псевдоним для открываемого объекта.
- **empty_layers** - Показывать пустые слои в общем списке `ProviderManager.read_contents()`

property id: `str`

Идентификатор провайдера.

is_supported_coordsystem(coordsystem: `CoordSystem`) → `bool`

Производится проверка, поддерживается ли СК провайдером.

Параметры

- **coordsystem** - Система Координат.

Пример:

```
coord_system = CoordSystem.from_prj('1,104')
print(f'Supported: {provider_manager.panorama.is_supported_coordsystem(coord_
↪system)}')
coord_system = CoordSystem.from_prj('32, 1020, 7, 42.5, 49.5, 78.5, 30.
↪28813972, 0, 0')
print(f'Supported: {provider_manager.panorama.is_supported_coordsystem(coord_
↪system)}')
>>> Supported: False
>>> Supported: True
```

open(filename: `Union[str, Path]`, alias: `Optional[str]` = None, empty_layers: `bool` = False) → `Table`

Открывает объект данных.

Параметры

- **filename** - Имя файла. Если это файлы *.sxf или *.txf, то их необходимо предварительно сконвертировать, используя метод `convert_file()`.
- **alias** - Псевдоним для открываемого объекта.

- `empty_layers` - Показывать пустые слои в общем списке`

22.1.29 OpenMode - Режим открытия файла

`class ахіру.OpenMode`

Режим открытия результирующего файла.

Атрибуты:

Create	Создание нового файла.
Append	Открытие существующего файла в режиме редактирования.

22.1.30 SqliteGeometryFormat - Формат данных sqlite

`class ахіру.SqliteGeometryFormat`

Формат данных sqlite. Используется при задании типа геометрии `SqliteDataProvider.get_destination()`

Атрибуты:

FGF	Формат FGF (FDO Geometry Format).
GPKG	Формат GPKG (GeoPackage).
SPATIALITE	Формат Spatialite.
WKB	Формат WKB (Well-Known Binary).
WKT	Формат WKT (Well-known text).

22.2 DataManager - Каталог данных

`class ахіру.DataManager`

Хранилище объектов данных. При открытии таблицы или растра эти объекты автоматически попадают в данный каталог. Для отслеживания изменений в каталоге используются события `added` и `removed`.

Примечание: Создание `ахіру.DataManager` не требуется, используйте объект `ахіру.data_manager`.

Список 11: Пример использования.

```
# Отслеживание добавления или удаления в каталоге.
data_manager.added[str].connect(lambda n: print(f'Таблица "{n}" добавлена в
↳ каталог'))
data_manager.removed[str].connect(lambda n: print(f'Таблица "{n}" удалена из
↳ каталога'))
# Отслеживание изменения в каталоге.
data_manager.updated.connect(lambda params: print(f"Каталог изменен: {params}"))
# Открываем таблицу
```

(continues on next page)

(продолжение с предыдущей страницы)

```

table = provider_manager.openfile(filepath)
# Список объектов каталога
for t in data_manager:
    print(t.name)
# Доступ по имени
'world' in data_manager
try:
    found_table = data_manager['world']
except KeyError:
    pass
# Закрываем таблицу
table.close()
# Убираем отслеживание
data_manager.added[str].disconnect()
data_manager.removed[str].disconnect()
'''
Таблица "world" добавлена в каталог
world
Таблица "world" удалена из каталога
'''

```

Свойства:

<code>all_objects</code>	Список всех объектов, включая скрытые.
<code>count</code>	Количество объектов данных.
<code>objects</code>	Список объектов.
<code>selection</code>	Таблица выборки, если она существует.
<code>sql_dialect</code>	Тип используемого диалекта по умолчанию для выполнения SQL-предложений.
<code>tables</code>	Список таблиц.

Методы:

<code>add(data_object)</code>	Добавляет объект данных в хранилище.
<code>check_query(query_text[, dialect])</code>	Производит проверку SQL-запроса на корректность.
<code>exists(obj)</code>	Проверяет, присутствует ли объект в каталоге.
<code>find(name)</code>	Производит поиск объект данных по имени.
<code>query(query_text[, dialect])</code>	Выполняет SQL-запрос к перечисленным таблицам.
<code>query_hidden(query_text[, dialect])</code>	Выполняет SQL-запрос к таблицам.
<code>remove(data_object)</code>	Удаляет объект данных.
<code>remove_all()</code>	Удаляет все объекты данных.

Сигналы:

<code>added</code>	Сигнал о добавлении объекта.
<code>removed</code>	Сигнал об удалении объекта.
<code>table_data_changed</code>	Сигнал об изменении данных таблиц, присутствующих в каталоге.
<code>updated</code>	Сигнал об изменении каталога.

Специальные методы:

<code>__contains__(name)</code>	Проверяет наличие элемента в объекте (оператор <code>in</code>).
<code>__getitem__(key)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__contains__(name: str) → bool`

Проверяет наличие элемента в объекте (оператор `in`).

`__getitem__(key: str) → DataObject`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[DataObject]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`add(data_object: DataObject)`

Добавляет объект данных в хранилище.

Параметры

`data_object` – Объект данных для добавления.

property `added`: `Signal`

Сигнал о добавлении объекта. В качестве параметра передается наименование таблицы.

Тип результата

`Signal[str]`

property `all_objects`: `List[DataObject]`

Список всех объектов, включая скрытые.

`check_query(query_text: str, dialect: TypeSqlDialect = TypeSqlDialect.sqlite) → Tuple[bool, str]`

Производит проверку SQL-запроса на корректность.

Параметры

- **`query_text`** – Текст запроса.
- **`dialect`** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как значение свойства `sql_dialect`.

Результат

Пара значений [Успешность проверки, Сообщение].

Список 12: Пример использования, если работа ведется в рамках Аксиома .

```
filepath = 'path/to/world.tab'
# Открываем таблицу
table = provider_manager.openfile(filepath)
table.name = 'world1'
# Проверка текста запроса
succ, mess = data_manager.check_query(f"select * from world1")
print(f'Check result: {succ}; {mess}')
'''
Check result: True; Запрос составлен верно
'''
```

property count: `int`

Количество объектов данных.

exists(obj: `DataObject`) → `bool`

Проверяет, присутствует ли объект в каталоге. Проверяет так-же и скрытые объекты, которые отсутствуют в общем списке.

Параметры

obj – проверяемый объект данных.

find(name: `str`) → `Optional[DataObject]`

Производит поиск объект данных по имени.

Параметры

name – Имя объекта данных.

Результат

Искомый объект данных или `None`.

property objects: `List[DataObject]`

Список объектов.

query(query_text: `str`, dialect: `TypeSqlDialect` = `TypeSqlDialect.sqlite`) → `Optional[Table]`

Выполняет SQL-запрос к перечисленным таблицам.

Параметры

- **query_text** – Текст запроса.
- **dialect** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как значение свойства `sql_dialect`.

Результат

Таблица, если результатом запроса является таблица.

Исключение

`RuntimeError` – При возникновении ошибки.

Список 13: Пример использования, если работа ведется в рамках Аксиома .

```
filepath = 'path/to/world.tab'
# Открываем таблицу
table = provider_manager.openfile(filepath)
table.name = 'world1'
# Выполняем запрос
qry = data_manager.query('select * from world1 where Страна like "A%")
# Выполняем тот-же запрос, но с явным указанием диалекта
qry = data_manager.query('select * from world1 where Страна like "A%"',
↳TypeSqlDialect.axioma)
```

query_hidden(query_text: str, dialect: TypeSqlDialect = TypeSqlDialect.sqlite) → Optional[Table]

Выполняет SQL-запрос к таблицам. В отличие от `query()` результирующий объект `Table` добавляется в каталог как скрытый объект. Он не учитывается в общем списке и от него из этого каталога не приходят события.

Параметры

- **query_text** – Текст запроса.
- **dialect** – Диалект, который используется при выполнении запроса. Значение по умолчанию установлено как `sql_dialect`.

Результат

Таблица, если результатом запроса является таблица.

remove(data_object: DataObject)

Удаляет объект данных.

Объект данных при этом закрывается.

Параметры

data_object – Объект данных для удаления.

remove_all()

Удаляет все объекты данных.

property removed: Signal

Сигнал об удалении объекта.

Тип результата

Signal[str]

property selection: Optional[SelectionTable]

Таблица выборки, если она существует.

См.также:

`axipy.selection_manager`

property sql_dialect: TypeSqlDialect

Тип используемого диалекта по умолчанию для выполнения SQL-предложений. Если необходимо переопределить, то для конкретного sql предложения необходимо указывать диалект явно `query()`.

Результат

Тип диалекта. Возможные значения `TypeSqlDialect.axioma` или `TypeSqlDialect.sqlite`.

property table_data_changed: Signal

Сигнал об изменении данных таблиц, присутствующих в каталоге. Испускается когда были изменены данные таблицы. В качестве параметров передаются имя изменяемой таблицы и дополнительная информация в виде dict (подробнее `Table.data_changed`).

Список 14: Пример подписки на изменение таблиц.

```
# Подписываемся на все присутствующие в менеджере таблицы
func = lambda name, info: print(f'Таблица {name} была изменена {info}')
data_manager.table_data_changed.connect(func)
# Отписаться от сигнала, если он больше не нужен
data_manager.table_data_changed.disconnect(func)
```

property tables: List[Table]

Список таблиц.

property updated: Signal

Сигнал об изменении каталога. В качестве параметра передается дополнительная информация в виде dict.

Таблица 3: Доступные параметры

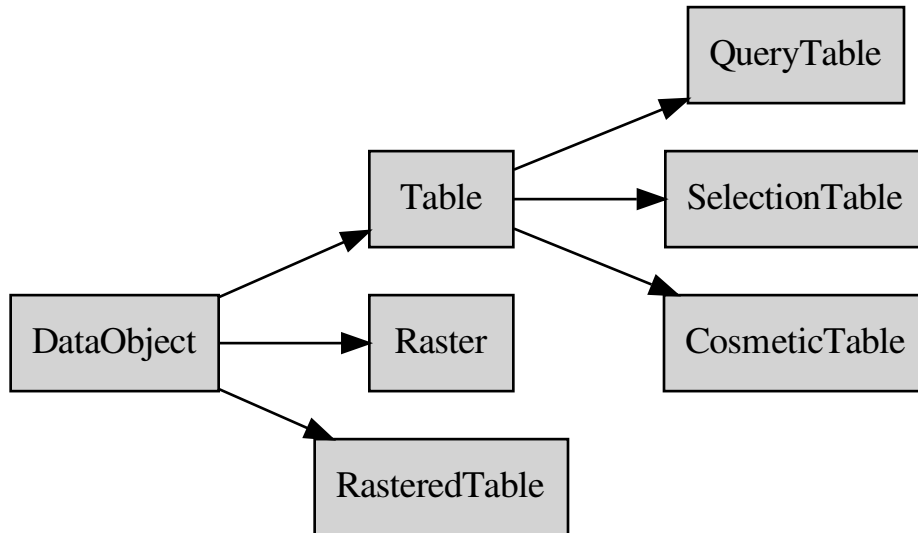
Наименование	Значение	Описание
operation	added	Таблица добавлена в каталог
operation	removed	Таблица удалена из каталога
operation	nameChanged	Изменено наименование таблицы
operation	selectionChanged	Произведены изменения в выборке
name		Наименование таблицы, если оно доступно

Тип результата

Signal[Dict[str, str]]

22.3 DataObject - Объект данных

Иерархия классов:



class ахіру.DataObject

Объект данных.

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

Пример:

```

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

Для закрытия объекта данных можно использовать менеджер контекста - выражение `with`. В таком случае таблица будет закрыта при выходе из блока. См. `close()`.

Пример:

```

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

Свойства:

<code>file_name</code>	Возвращает путь к файлу таблицы.
<code>is_spatial</code>	Признак того, что объект данных является пространственным.
<code>is_valid</code>	Возвращает признак валидности объекта.
<code>name</code>	Название объекта данных.
<code>properties</code>	Дополнительные свойства объекта данных.
<code>provider</code>	Провайдер изначального источника данных.

Методы:

<code>close()</code>	Пытается закрыть таблицу.

Сигналы:

<code>destroyed</code>	Сигнал оповещения об удалении объекта.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__() → DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object) → bool`

Оператор равенства (`==`).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType]) → Optional[bool]`

Закрывает таблицу при выходе из блока кода `with`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`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 destroyed: `Signal`

Сигнал оповещения об удалении объекта.

property file_name: `Optional[Path]`

Возвращает путь к файлу таблицы.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_valid: `bool`

Возвращает признак валидности объекта.

property name: `str`

Название объекта данных.

property properties: `dict`

Дополнительные свойства объекта данных.

property provider: `str`

Провайдер изначального источника данных.

22.4 Table - Таблица

class `axipy.Table`

Базовые классы: `DataObject`

Таблица.

Менеджер контекста сохраняет изменения и закрывает таблицу.

Пример:

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

См.также:

`commit()`, `DataObject.close()`.

Свойства:

<code>can_redo</code>	Возможен ли откат на один шаг вперед.
<code>can_undo</code>	Возможен ли откат на один шаг назад.
<code>coordsystem</code>	Система координат таблицы.
<code>file_name</code>	Возвращает путь к файлу таблицы.
<code>hotlink</code>	Наименование атрибута таблицы для хранения гиперссылки.
<code>is_editable</code>	Признак того, что таблица является редактируемой.
<code>is_modified</code>	Таблица содержит несохраненные изменения.
<code>is_spatial</code>	Признак того, что объект данных является пространственным.
<code>is_temporary</code>	Признак того, что таблица является временной.
<code>is_valid</code>	Возвращает признак валидности объекта.
<code>name</code>	Название объекта данных.
<code>properties</code>	Дополнительные свойства объекта данных.
<code>provider</code>	Провайдер изначального источника данных.
<code>schema</code>	Схема (структура) таблицы.
<code>supported_operations</code>	Доступные операции.

Методы:

<code>class_geometries()</code>	Возвращает перечень геометрических типов <code>GeometryClass</code> , содержащихся в таблице.
<code>close()</code>	Пытается закрыть таблицу.
<code>commit()</code>	Сохраняет изменения в таблице.
<code>count([bbox])</code>	Возвращает количество записей, удовлетворяющих параметрам.
<code>get_bounds()</code>	Возвращает область, в которую попадают все данные таблицы.
<code>insert(features)</code>	Вставляет записи в таблицу.
<code>items([bbox, ids])</code>	Запрашивает записи, удовлетворяющие параметрам.
<code>itemsByIds(ids)</code>	Запрашивает записи по списку <code>list</code> с идентификаторами записей, либо перечень идентификаторов в виде списка.
<code>itemsInObject(obj)</code>	Запрашивает записи с фильтром по геометрическому объекту.
<code>itemsInRect(bbox)</code>	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
<code>redo([steps])</code>	Производит откат вперед на заданное количество шагов.
<code>remove(ids)</code>	Удаляет записи из таблицы.
<code>rollback()</code>	Отменяет несохраненные изменения в таблице.
<code>undo([steps])</code>	Производит откат назад на заданное количество шагов.
<code>update(features)</code>	Обновляет записи в таблице.

Сигналы:

<code>data_changed</code>	Сигнал об изменении данных таблицы.
<code>destroyed</code>	Сигнал оповещения об удалении объекта.
<code>schema_changed</code>	Сигнал об изменении схемы таблицы.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__()` → `DataObject`

Вход в контекст (используется с оператором `with`).

__eq__(other: `object`) → `bool`

Оператор равенства (==).

__exit__(exception_type: `Optional[Type[BaseException]]`, exception_value: `Optional[BaseException]`, traceback: `Optional[TracebackType]`) → `Optional[bool]`

Закрывает таблицу при выходе из блока кода with.

__iter__() → `Iterator[Feature]`

Возвращает итератор по объекту (функция `iter()`).

__len__() → `int`

Длина объекта (функция `len()`).

__repr__() → `str`

Официальное строковое представление объекта при вызове `repr()`.

property can_redo: `bool`

Возможен ли откат на один шаг вперед.

property can_undo: `bool`

Возможен ли откат на один шаг назад.

class geometries() → `List[GeometryClass]`

Возвращает перечень геометрических типов `GeometryClass`, содержащихся в таблице.

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`.

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение

RuntimeError – Невозможно сохранить изменения.

property coordsystem: `CoordSystem`

Система координат таблицы.

count(bbox: `Optional[Union[Rect, QRectF, tuple]]` = None) → `int`

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 – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: Signal

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 4: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
ids	list	Перечень затронутых идентификаторов, если он доступен. Если идентификаторы имеют отрицательные значения, то они относятся к новым несохраненным записям.

См.также:

`DataManager.table_data_changed`

Список 15: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
# Подписываемся на конкретную таблицу
func = lambda info: print(f'Таблица была изменена {info}')
table.data_changed.connect(func)
```

property destroyed: Signal

Сигнал оповещения об удалении объекта.

property file_name: Optional[Path]

Возвращает путь к файлу таблицы.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 5: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.VectorLayer.hotlink`.

insert (features: `Union[Feature, Iterable[Feature]]`)

Вставляет записи в таблицу.

Параметры

features – Записи для вставки.

property is_editable: `bool`

Признак того, что таблица является редактируемой.

property is_modified: `bool`

Таблица содержит несохраненные изменения.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_temporary: `bool`

Признак того, что таблица является временной.

Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

property is_valid: `bool`

Возвращает признак валидности объекта.

items (bbox: `Optional[Union[Rect, QRectF, tuple]]` = None, ids: `Optional[List[int]]` = None) → `Iterator[Feature]`

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат

Итератор по записям.

itemsByIds (ids: `List[int]`) → `Iterator[Feature]`

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры

ids – Список идентификаторов.

Результат

Итератор по записям.

Список 16: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры

obj - Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат

Итератор по записям.

Список 17: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
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: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры

bbox - Ограничивающий прямоугольник.

Результат

Итератор по записям.

Список 18: Пример запроса (таблица в проекции Робинсона)

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

property name: **str**

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

remove(ids: Union[int, List[int]])

Удаляет записи из таблицы.

Параметры

ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема (структура) таблицы.

Если требуется изменить схему существующей таблицы, необходимо учесть следующие моменты:

- Изменение производится напрямую в таблице. Необходимо позаботиться о создании резервной копии.
- Перед изменением нужно сохранить все данные. В противном случае при попытке изменения таблицы с несохраненными данными будет вызвано исключение.

Список 19: Пример изменения существующей схемы таблицы

```
# Открываем таблицу
table_world = provider_manager.openfile(filepath)
# Текущая схема таблицы
schema = table_world.schema
# Удаление атрибута
schema.delete('Население')
# Обновление атрибута (изменение его имени)
schema['Мужское'] = Attribute.string('Мужское_население', 30)
# Добавление атрибута
schema.append(Attribute.string('Новый атрибут', 30))
# Обновляем схему таблицы
table_world.schema = schema
```

См.также:

[Schema](#)

property schema_changed: Signal

Сигнал об изменении схемы таблицы.

Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 20: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат назад на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры

features – Записи для обновления.

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

Список 21: Пример использования

```
import axipy

table: axipy.Table
target_id = 1

items = table.itemsByIds([target_id])
feature = next(items)
feature["attr_name"] = "new_value"
```

См.также:

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

22.5 QueryTable - SQL запрос.

class axipy.QueryTable

Базовые классы: `Table`

Таблица, построенная на основе SQL запроса.

Пример:

```
table = provider_manager.openfile('world.tab')
query = data_manager.query('select * from world')
```

Свойства:

<code>can_redo</code>	Возможен ли откат на один шаг вперед.
<code>can_undo</code>	Возможен ли откат на один шаг назад.
<code>coordsystem</code>	Система координат таблицы.
<code>file_name</code>	Возвращает путь к файлу таблицы.
<code>hotlink</code>	Наименование атрибута таблицы для хранения гиперссылки.
<code>is_editable</code>	Признак того, что таблица является редактируемой.
<code>is_modified</code>	Таблица содержит несохраненные изменения.
<code>is_spatial</code>	Признак того, что объект данных является пространственным.
<code>is_temporary</code>	Признак того, что таблица является временной.
<code>is_valid</code>	Возвращает признак валидности объекта.
<code>name</code>	Название объекта данных.
<code>properties</code>	Дополнительные свойства объекта данных.
<code>provider</code>	Провайдер изначального источника данных.
<code>schema</code>	Схема (структура) таблицы.
<code>sql_text</code>	Текст SQL запроса.
<code>supported_operations</code>	Доступные операции.

Методы:

<code>class_geometries()</code>	Возвращает перечень геометрических типов <code>GeometryClass</code> , содержащихся в таблице.
<code>close()</code>	Пытается закрыть таблицу.
<code>commit()</code>	Сохраняет изменения в таблице.
<code>count([bbox])</code>	Возвращает количество записей, удовлетворяющих параметрам.
<code>get_bounds()</code>	Возвращает область, в которую попадают все данные таблицы.
<code>insert(features)</code>	Вставляет записи в таблицу.
<code>items([bbox, ids])</code>	Запрашивает записи, удовлетворяющие параметрам.
<code>itemsByIds(ids)</code>	Запрашивает записи по списку <code>list</code> с идентификаторами записей, либо перечень идентификаторов в виде списка.
<code>itemsInObject(obj)</code>	Запрашивает записи с фильтром по геометрическому объекту.
<code>itemsInRect(bbox)</code>	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
<code>redo([steps])</code>	Производит откат вперед на заданное количество шагов.
<code>remove(ids)</code>	Удаляет записи из таблицы.
<code>rollback()</code>	Отменяет несохраненные изменения в таблице.
<code>undo([steps])</code>	Производит откат назад на заданное количество шагов.
<code>update(features)</code>	Обновляет записи в таблице.

Сигналы:

<code>data_changed</code>	Сигнал об изменении данных таблицы.
<code>destroyed</code>	Сигнал оповещения об удалении объекта.
<code>schema_changed</code>	Сигнал об изменении схемы таблицы.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__()` → `DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType]) → Optional[bool]`

Закрывает таблицу при выходе из блока кода with.

`__iter__() → Iterator[Feature]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

class geometries() → List[GeometryClass]

Возвращает перечень геометрических типов `GeometryClass`, содержащихся в таблице.

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`.

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение

RuntimeError – Невозможно сохранить изменения.

property coordsystem: CoordSystem

Система координат таблицы.

count(bbox: Optional[Union[Rect, QRectF, tuple]] = None) → int

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 – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: Signal

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 6: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
ids	list	Перечень затронутых идентификаторов, если он доступен. Если идентификаторы имеют отрицательные значения, то они относятся к новым несохраненным записям.

См.также:

`DataManager.table_data_changed`

Список 22: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
# Подписываемся на конкретную таблицу
func = lambda info: print(f'Таблица была изменена {info}')
table.data_changed.connect(func)
```

property destroyed: Signal

Сигнал оповещения об удалении объекта.

property file_name: Optional[Path]

Возвращает путь к файлу таблицы.

get_bounds() → Rect

Возвращает область, в которую попадают все данные таблицы.

property hotlink: str

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 7: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.VectorLayer.hotlink`.

insert (features: `Union[Feature, Iterable[Feature]]`)

Вставляет записи в таблицу.

Параметры

features – Записи для вставки.

property is_editable: `bool`

Признак того, что таблица является редактируемой.

property is_modified: `bool`

Таблица содержит несохраненные изменения.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_temporary: `bool`

Признак того, что таблица является временной.

Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

property is_valid: `bool`

Возвращает признак валидности объекта.

items (bbox: `Optional[Union[Rect, QRectF, tuple]]` = None, ids: `Optional[List[int]]` = None) → `Iterator[Feature]`

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат

Итератор по записям.

itemsByIds (ids: `List[int]`) → `Iterator[Feature]`

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры

ids – Список идентификаторов.

Результат

Итератор по записям.

Список 23: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры

obj - Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат

Итератор по записям.

Список 24: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
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: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры

bbox - Ограничивающий прямоугольник.

Результат

Итератор по записям.

Список 25: Пример запроса (таблица в проекции Робинсона)

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

property name: **str**

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

remove(ids: Union[int, List[int]])

Удаляет записи из таблицы.

Параметры

ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема (структура) таблицы.

Если требуется изменить схему существующей таблицы, необходимо учесть следующие моменты:

- Изменение производится напрямую в таблице. Необходимо позаботиться о создании резервной копии.
- Перед изменением нужно сохранить все данные. В противном случае при попытке изменения таблицы с несохраненными данными будет вызвано исключение.

Список 26: Пример изменения существующей схемы таблицы

```
# Открываем таблицу
table_world = provider_manager.openfile(filepath)
# Текущая схема таблицы
schema = table_world.schema
# Удаление атрибута
schema.delete('Население')
# Обновление атрибута (изменение его имени)
schema['Мужское'] = Attribute.string('Мужское_население', 30)
# Добавление атрибута
schema.append(Attribute.string('Новый атрибут', 30))
# Обновляем схему таблицы
table_world.schema = schema
```

См.также:

[Schema](#)

property schema_changed: `Signal`

Сигнал об изменении схемы таблицы.

Испускается когда была изменена структура таблицы.

property sql_text: `str`

Текст SQL запроса.

property supported_operations: `SupportedOperations`

Доступные операции.

Список 27: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: `int` = 1)

Производит откат назад на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

update(features: `Union[Feature, Iterable[Feature]]`)

Обновляет записи в таблице.

Параметры

features – Записи для обновления.

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

Список 28: Пример использования

```
import axipy

table: axipy.Table
target_id = 1

items = table.itemsByIds([target_id])
feature = next(items)
feature["attr_name"] = "new_value"
```

См.также:

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

22.6 SelectionTable - Таблица с текущей выборкой.

class axipy.SelectionTable

Базовые классы: `Table`

Таблица, построенная на основе текущей выборки. Носит временный характер. Создается, когда в выборку добавляются объекты и удаляется, когда выборка очищается.

Пример доступа к выборке с получением идентификаторов:

```
for f in data_manager.selection.items():
    print('>>>', f.id)
```

Свойства:

base_table	Таблица, на которой основана данная выборка.
can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
file_name	Возвращает путь к файлу таблицы.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
is_valid	Возвращает признак валидности объекта.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема (структура) таблицы.
supported_operations	Доступные операции.

Методы:

<code>class_geometries()</code>	Возвращает перечень геометрических типов <code>GeometryClass</code> , содержащихся в таблице.
<code>close()</code>	Пытается закрыть таблицу.
<code>commit()</code>	Сохраняет изменения в таблице.
<code>count([bbox])</code>	Возвращает количество записей, удовлетворяющих параметрам.
<code>get_bounds()</code>	Возвращает область, в которую попадают все данные таблицы.
<code>insert(features)</code>	Вставляет записи в таблицу.
<code>items([bbox, ids])</code>	Запрашивает записи, удовлетворяющие параметрам.
<code>itemsByIds(ids)</code>	Запрашивает записи по списку <code>list</code> с идентификаторами записей, либо перечень идентификаторов в виде списка.
<code>itemsInObject(obj)</code>	Запрашивает записи с фильтром по геометрическому объекту.
<code>itemsInRect(bbox)</code>	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
<code>redo([steps])</code>	Производит откат вперед на заданное количество шагов.
<code>remove(ids)</code>	Удаляет записи из таблицы.
<code>rollback()</code>	Отменяет несохраненные изменения в таблице.
<code>undo([steps])</code>	Производит откат назад на заданное количество шагов.
<code>update(features)</code>	Обновляет записи в таблице.

Сигналы:

<code>data_changed</code>	Сигнал об изменении данных таблицы.
<code>destroyed</code>	Сигнал оповещения об удалении объекта.
<code>schema_changed</code>	Сигнал об изменении схемы таблицы.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__()` → `DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType]) → Optional[bool]`

Закрывает таблицу при выходе из блока кода with.

`__iter__() → Iterator[Feature]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property `base_table: Table`

Таблица, на которой основана данная выборка.

property `can_redo: bool`

Возможен ли откат на один шаг вперед.

property `can_undo: bool`

Возможен ли откат на один шаг назад.

class `geometries() → List[GeometryClass]`

Возвращает перечень геометрических типов `GeometryClass`, содержащихся в таблице.

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`.

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение

`RuntimeError` – Невозможно сохранить изменения.

property `coordsystem: CoordSystem`

Система координат таблицы.

count(bbox: Optional[Union[Rect, QRectF, tuple]] = None) → int

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 – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: **Signal**

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 8: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
ids	list	Перечень затронутых идентификаторов, если он доступен. Если идентификаторы имеют отрицательные значения, то они относятся к новым несохраненным записям.

См.также:

`DataManager.table_data_changed`

Список 29: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
# Подписываемся на конкретную таблицу
func = lambda info: print(f'Таблица была изменена {info}')
table.data_changed.connect(func)
```

property destroyed: **Signal**

Сигнал оповещения об удалении объекта.

property file_name: **Optional[Path]**

Возвращает путь к файлу таблицы.

get_bounds() → **Rect**

Возвращает область, в которую попадают все данные таблицы.

property hotlink: `str`

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 9: Возможны следующие варианты

Значение	Описание
<code>axioma://world.tab</code>	Открывает файл или рабочее пространство в аксиоме
<code>addlayer://world</code>	Добавляет слой world в текущую карту
<code>exec://gimp</code>	Запускает на выполнение программу gimp
<code>https://axioma-gis.ru/</code>	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.VectorLayer.hotlink`.

insert (features: `Union[Feature, Iterable[Feature]]`)

Вставляет записи в таблицу.

Параметры

features – Записи для вставки.

property is_editable: `bool`

Признак того, что таблица является редактируемой.

property is_modified: `bool`

Таблица содержит несохраненные изменения.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_temporary: `bool`

Признак того, что таблица является временной.

Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

property is_valid: `bool`

Возвращает признак валидности объекта.

items (bbox: `Optional[Union[Rect, QRectF, tuple]]` = None, ids: `Optional[List[int]]` = None) → `Iterator[Feature]`

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат

Итератор по записям.

itemsByIds (ids: `List[int]`) → `Iterator[Feature]`

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры

ids – Список идентификаторов.

Результат

Итератор по записям.

Список 30: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры

obj – Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат

Итератор по записям.

Список 31: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
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: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры

bbox – Ограничивающий прямоугольник.

Результат

Итератор по записям.

Список 32: Пример запроса (таблица в проекции Робинсона)

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

property name: `str`

Название объекта данных.

property properties: `dict`

Дополнительные свойства объекта данных.

property provider: `str`

Провайдер изначального источника данных.

redo(steps: `int` = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

remove(ids: `Union[int, List[int]]`)

Удаляет записи из таблицы.

Параметры

ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: `Schema`

Схема (структура) таблицы.

Если требуется изменить схему существующей таблицы, необходимо учесть следующие моменты:

- Изменение производится напрямую в таблице. Необходимо позаботиться о создании резервной копии.
- Перед изменением нужно сохранить все данные. В противном случае при попытке изменения таблицы с несохраненными данными будет вызвано исключение.

Список 33: Пример изменения существующей схемы таблицы

```
# Открываем таблицу
table_world = provider_manager.openfile(filepath)
# Текущая схема таблицы
schema = table_world.schema
# Удаление атрибута
schema.delete('Население')
# Обновление атрибута (изменение его имени)
schema['Мужское'] = Attribute.string('Мужское_население', 30)
# Добавление атрибута
schema.append(Attribute.string('Новый атрибут', 30))
# Обновляем схему таблицы
table_world.schema = schema
```

См.также:

`Schema`

property schema_changed: Signal

Сигнал об изменении схемы таблицы.

Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 34: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат назад на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры

features – Записи для обновления.

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

Список 35: Пример использования

```
import axipy

table: axipy.Table
target_id = 1

items = table.itemsByIds([target_id])
feature = next(items)
feature["attr_name"] = "new_value"
```

См.также:

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

22.7 CosmeticTable - Таблица с данными косметического слоя.

class axipy.CosmeticTable

Базовые классы: `Table`

Объект данных косметического слоя `axipy.CosmeticLayer`

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
coordsystem	Система координат таблицы.
file_name	Возвращает путь к файлу таблицы.
hotlink	Наименование атрибута таблицы для хранения гиперссылки.
is_editable	Признак того, что таблица является редактируемой.
is_modified	Таблица содержит несохраненные изменения.
is_spatial	Признак того, что объект данных является пространственным.
is_temporary	Признак того, что таблица является временной.
is_valid	Возвращает признак валидности объекта.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема (структура) таблицы.
supported_operations	Доступные операции.

Методы:

<code>class_geometries()</code>	Возвращает перечень геометрических типов <code>GeometryClass</code> , содержащихся в таблице.
<code>close()</code>	Пытается закрыть таблицу.
<code>commit()</code>	Сохраняет изменения в таблице.
<code>count([bbox])</code>	Возвращает количество записей, удовлетворяющих параметрам.
<code>get_bounds()</code>	Возвращает область, в которую попадают все данные таблицы.
<code>insert(features)</code>	Вставляет записи в таблицу.
<code>items([bbox, ids])</code>	Запрашивает записи, удовлетворяющие параметрам.
<code>itemsByIds(ids)</code>	Запрашивает записи по списку <code>list</code> с идентификаторами записей, либо перечень идентификаторов в виде списка.
<code>itemsInObject(obj)</code>	Запрашивает записи с фильтром по геометрическому объекту.
<code>itemsInRect(bbox)</code>	Запрашивает записи с фильтром по ограничивающему прямоугольнику.
<code>redo([steps])</code>	Производит откат вперед на заданное количество шагов.
<code>remove(ids)</code>	Удаляет записи из таблицы.
<code>rollback()</code>	Отменяет несохраненные изменения в таблице.
<code>undo([steps])</code>	Производит откат назад на заданное количество шагов.
<code>update(features)</code>	Обновляет записи в таблице.

Сигналы:

<code>data_changed</code>	Сигнал об изменении данных таблицы.
<code>destroyed</code>	Сигнал оповещения об удалении объекта.
<code>schema_changed</code>	Сигнал об изменении схемы таблицы.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__()` → `DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType]) → Optional[bool]`

Закрывает таблицу при выходе из блока кода with.

`__iter__() → Iterator[Feature]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

class geometries() → List[GeometryClass]

Возвращает перечень геометрических типов `GeometryClass`, содержащихся в таблице.

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`.

commit()

Сохраняет изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

Исключение

RuntimeError – Невозможно сохранить изменения.

property coordsystem: CoordSystem

Система координат таблицы.

count(bbox: Optional[Union[Rect, QRectF, tuple]] = None) → int

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 – Ограничивающий прямоугольник.

Результат

Количество записей.

property data_changed: **Signal**

Сигнал об изменении данных таблицы. Испускается когда были изменены данные таблицы. В качестве параметра передается дополнительная информация в виде dict.

Таблица 10: Доступные параметры

Наименование	Значение	Описание
operation	insert	Произведена вставка данных
operation	update	Данные были обновлены
operation	remove	Данные были удалены
ids	list	Перечень затронутых идентификаторов, если он доступен. Если идентификаторы имеют отрицательные значения, то они относятся к новым несохраненным записям.

См.также:

`DataManager.table_data_changed`

Список 36: Пример подписки на изменение таблицы.

```
table = provider_manager.openfile(filepath)
# Подписываемся на конкретную таблицу
func = lambda info: print(f'Таблица была изменена {info}')
table.data_changed.connect(func)
```

property destroyed: **Signal**

Сигнал оповещения об удалении объекта.

property file_name: **Optional[Path]**

Возвращает путь к файлу таблицы.

get_bounds() → **Rect**

Возвращает область, в которую попадают все данные таблицы.

property hotlink: **str**

Наименование атрибута таблицы для хранения гиперссылки.

Таблица 11: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

См.также:

`axipy.VectorLayer.hotlink`.

insert (features: `Union[Feature, Iterable[Feature]]`)

Вставляет записи в таблицу.

Параметры

features – Записи для вставки.

property is_editable: `bool`

Признак того, что таблица является редактируемой.

property is_modified: `bool`

Таблица содержит несохраненные изменения.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_temporary: `bool`

Признак того, что таблица является временной.

Это в первую очередь касается таблиц, созданных в памяти. А также таблица косметического слоя.

property is_valid: `bool`

Возвращает признак валидности объекта.

items (bbox: `Optional[Union[Rect, QRectF, tuple]]` = None, ids: `Optional[List[int]]` = None) → `Iterator[Feature]`

Запрашивает записи, удовлетворяющие параметрам. В качестве фильтра может быть указан либо ограничивающий прямоугольник, либо перечень идентификаторов в виде списка.

Параметры

- **bbox** – Ограничивающий прямоугольник.
- **ids** – Список идентификаторов.

Результат

Итератор по записям.

itemsByIds (ids: `List[int]`) → `Iterator[Feature]`

Запрашивает записи по списку `list` с идентификаторами записей, либо перечень идентификаторов в виде списка. Идентификаторы несохраненных записей имеют отрицательные значения.

Параметры

ids – Список идентификаторов.

Результат

Итератор по записям.

Список 37: Пример

```
table_world = provider_manager.openfile(filepath)
# Пример запроса по списку идентификаторов.
items = table_world.itemsByIds([11, 27, 41, 163, 203])
for f in items:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
# Просмотр идентификаторов всех записей, включая несохраненные
for f in table_world.items():
    print(f.id, f['Страна'])
# Получение несохраненных записей совместно с сохраненными
items_new = table_world.itemsByIds([-1, -12, 27])
for f in items_new:
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

itemsInObject(obj: Geometry) → Iterator[Feature]

Запрашивает записи с фильтром по геометрическому объекту.

Параметры

obj - Геометрия. Если для нее не задана СК, используется СК таблицы.

Результат

Итератор по записям.

Список 38: Пример запроса по полигону

```
table_world = provider_manager.openfile(filepath)
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: Union[Rect, QRectF, tuple]) → Iterator[Feature]

Запрашивает записи с фильтром по ограничивающему прямоугольнику.

Параметры

bbox - Ограничивающий прямоугольник.

Результат

Итератор по записям.

Список 39: Пример запроса (таблица в проекции Робинсона)

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

property name: **str**

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

redo(steps: int = 1)

Производит откат вперед на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

remove(ids: Union[int, List[int]])

Удаляет записи из таблицы.

Параметры

ids – Идентификаторы записей для удаления.

rollback()

Отменяет несохраненные изменения в таблице.

Если таблица не содержит несохраненные изменения, то команда игнорируется.

property schema: Schema

Схема (структура) таблицы.

Если требуется изменить схему существующей таблицы, необходимо учесть следующие моменты:

- Изменение производится напрямую в таблице. Необходимо позаботиться о создании резервной копии.
- Перед изменением нужно сохранить все данные. В противном случае при попытке изменения таблицы с несохраненными данными будет вызвано исключение.

Список 40: Пример изменения существующей схемы таблицы

```
# Открываем таблицу
table_world = provider_manager.openfile(filepath)
# Текущая схема таблицы
schema = table_world.schema
# Удаление атрибута
schema.delete('Население')
# Обновление атрибута (изменение его имени)
schema['Мужское'] = Attribute.string('Мужское_население', 30)
# Добавление атрибута
schema.append(Attribute.string('Новый атрибут', 30))
# Обновляем схему таблицы
table_world.schema = schema
```

См.также:

[Schema](#)

property schema_changed: Signal

Сигнал об изменении схемы таблицы.

Испускается когда была изменена структура таблицы.

property supported_operations: SupportedOperations

Доступные операции.

Список 41: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

undo(steps: int = 1)

Производит откат назад на заданное количество шагов. При этом возвращается состояние до последней отмены.

Параметры

steps – Количество шагов.

update(features: Union[Feature, Iterable[Feature]])

Обновляет записи в таблице.

Параметры

features – Записи для обновления.

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

Список 42: Пример использования

```
import axipy

table: axipy.Table
target_id = 1

items = table.itemsByIds([target_id])
feature = next(items)
feature["attr_name"] = "new_value"
```

См.также:

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

22.8 SupportedOperations - Доступные операции

class axipy.SupportedOperations

Флаги доступных операций.

Реализованы как перечисление `enum.IntFlag` и поддерживают побитовые операции.

Атрибут	Значение
Empty	0
Insert	1
Update	2
Delete	4
Write	Insert Update Delete = 7
Read	8
ReadWrite	Read Write = 15

Список 43: Пример использования

```
flags = table.supported_operations
assert flags == SupportedOperations.ReadWrite
assert flags & SupportedOperations.Insert
assert SupportedOperations.Write in flags
```

См.также:

`enum.IntFlag`, `axipy.Table.supported_operations`

22.9 Feature - Запись в таблице

class axipy.Feature

Запись в таблице.

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

Список 44: Примеры.

```
feature = Feature({'attr_name': 'value'}, geometry=Point(10, 10),
style=PointStyle.create_mi_compat(35, 0))
# Количество атрибутов
count = len(feature)
# Запись значения по ключу
feature['attr_name'] = 'new_value'
# Запись значения по индексу
feature[0] = 'another_value'
# Чтение значения по ключу
value = feature['attr_name']
# Чтение значения по индексу
another_value = feature[0]
# Проверка наличия атрибута по ключу
'attr_name' in feature
# Проверка наличия атрибута по индексу
5 in feature
# Значения атрибутов можно задать словарем или именованными аргументами:
feature2 = Feature({'name1': 'value1', 'name2': 'value2'})
# Это эквивалентно
feature2 = Feature(name1='value1', name2='value2')
# Получение стиля оформления для геометрии
style = feature.style
# Установка нового стиля для геометрии
```

(continues on next page)

(продолжение с предыдущей страницы)

```
feature.style = style
# Получение геометрии
point = feature.geometry
# Установка нового значения для геометрии
feature.geometry = Point(20, 20)
# Просмотр всех наименований и значений атрибутов
for key, value in feature.items():
    print('{} = {}'.format(key, value))
...

>>> attr_name = value
>>> +geometry = Point pos=(10.0 10.0)
>>> +style = PointStyle Symbol (35, 0, 8)
...
```

Примечание: Для доступа к геометрическому атрибуту и стилю по наименованию можно использовать predefined идентификаторы +geometry и +style соответственно:

- GEOMETRY_ATTR=+geometry
- STYLE_ATTR=+style

Конструктор класса:

```
__init__([properties, geometry, style, Конструктор класса.
id])
```

Свойства:

geometry	Устанавливает или возвращает геометрию записи.
id	Устанавливает или возвращает идентификатор записи в таблице.
style	Устанавливает или возвращает стиль записи.

Методы:

get(key[, default])	Возвращает значение заданного атрибута.
has_geometry()	Проверяет, имеет ли запись атрибут с геометрией.
has_style()	Проверяет, имеет ли запись атрибут со стилем.
items()	Возвращает список пар имя — значение.
keys()	Возвращает список имен атрибутов.
to_geojson()	Представляет запись в виде, похожем на 'GeoJSON'.
values()	Возвращает список значений атрибутов.

Специальные методы:

<code>__contains__(key)</code>	Проверяет наличие атрибута у записи.
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__getitem__(index)</code>	Возвращает значение атрибута по индексу или ключу.
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Возвращает количество атрибутов в записи.
<code>__setitem__(index, value)</code>	Присваивает атрибуту значение по индексу или ключу.

`__contains__(key: Union[int, str]) → bool`

Проверяет наличие атрибута у записи.

Параметры

key - Имя или номер атрибута.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__getitem__(index: Union[int, str]) → Any`

Возвращает значение атрибута по индексу или ключу.

Параметры

index - Индекс атрибута или его имя.

`__init__(properties: dict = dict(), geometry: Optional[Geometry] = None, style: Optional[Style] = None, id: Optional[int] = None, **kwargs: Any)`

Конструктор класса.

Параметры

- **properties** - Значения атрибутов.
- **geometry** - Геометрия.
- **style** - Стил.
- **id** - Идентификатор.
- ****kwargs** - Значения атрибутов.

`__iter__() → Iterator[Any]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Возвращает количество атрибутов в записи.

`__setitem__(index: Union[int, str], value: Any)`

Присваивает атрибуту значение по индексу или ключу.

Параметры

- **index** - индекс атрибута или его имя.
- **value** - Присваиваемое значение.

property geometry: `Optional[Geometry]`

Устанавливает или возвращает геометрию записи.

См.также:

`Feature.has_geometry()`, `GEOMETRY_ATTR`

Результат

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

get(key: `str`, default: `Optional[Any] = None`) → `Any`

Возвращает значение заданного атрибута.

Параметры

- **key** - Имя атрибута.
- **default** - Значение по умолчанию.

Результат

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

has_geometry() → `bool`

Проверяет, имеет ли запись атрибут с геометрией.

has_style() → `bool`

Проверяет, имеет ли запись атрибут со стилем.

property id: `int`

Устанавливает или возвращает идентификатор записи в таблице.

Несохраненные записи в таблице будут иметь отрицательное значение.

См.также:

`Table.is_modified`, `Table.commit()`

Результат

0 если идентификатор не задан.

items() → `List[Tuple[str, Any]]`

Возвращает список пар имя — значение.

keys() → `List[str]`

Возвращает список имен атрибутов.

property style: `Optional[Style]`

Устанавливает или возвращает стиль записи.

См.также:

`Feature.has_style()`, `STYLE_ATTR`

Результат

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

`to_geojson()` → dict

Представляет запись в виде, похожем на „GeoJSON“.

`values()` → List[Any]

Возвращает список значений атрибутов.

22.10 Schema - Схема таблицы

class axipy.Schema

Базовые классы: list

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

Список 45: Пример создания

```
# Определяем схему таблицы
schema = Schema(
    Attribute.string('one', 25),
    Attribute.integer('two'),
    Attribute.decimal('three'),
    coordsystem='prj:Earth Projection 12, 62, "m", 0'
)
# На базе этой схемы можно создать таблицу
definition = {
    'src': '',
    'schema': schema
}
table = provider_manager.create(definition)
```

Список 46: Пример создания из списка

```
attrs = [Attribute.string('one', 25), Attribute.integer(
    'two'), Attribute.decimal('three')]
# распаковывается список атрибутов
schema = Schema(*attrs, coordsystem='prj:Earth Projection 12, 62, "m", 0')
```

Имеет стандартные функции работы со списком.

Список 47: Пример операций

```
# Вставка атрибута на 2 позицию
schema.insert(2, Attribute.string('attr_name'))
# Количество атрибутов
count = len(schema)
# Получение атрибута по индексу
attribute = schema[2]
# Получение атрибута по имени
attribute = schema['attr_name']
# Изменение атрибута по индексу
schema[2] = Attribute.string('attr', 30)
# Изменение атрибута по имени
schema['name_attribute'] = Attribute.string('attr', 30)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Удаление атрибута по его индексу
del schema[2]
# Аналогичная команда удаления атрибута
schema.delete(2)
# Удаление атрибута по имени
schema.delete('Население')
# Добавление атрибута в конец
schema.append(Attribute.string('new_attr'))
# Вставка по индексу
schema.insert(2, Attribute.integer('num_attr'))
# Поиск атрибута по имени
index = schema.index('new_attr')
# проверяет существование
exists = 'new_attr' in schema
# Удаление по имени
schema.remove('new_attr')
# Получение атрибута по его имени
a = schema.by_name('attr')
```

См.также:

Изменение структуры существующей таблицы см. [Table.schema](#)

Конструктор класса:

<code>__init__(*attributes[, coordsystem])</code>	Конструктор класса.

Свойства:

<code>attribute_names</code>	Возвращает список имен атрибутов.
<code>coordsystem</code>	Устанавливает или возвращает систему координат.

Методы:

<code>by_name(n)</code>	Возвращает атрибут по его имени.
<code>delete(index)</code>	Удаляет атрибут.
<code>index_by_name(n)</code>	Возвращает индекс по имени атрибута.
<code>insert(index, attr)</code>	Вставляет атрибут.

Специальные методы:

<code>__contains__(name)</code>	Проверяет наличие элемента в объекте (оператор <code>in</code>).
<code>__getitem__(index)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__setitem__(index, value)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).

`__contains__(name: object) → bool`

Проверяет наличие элемента в объекте (оператор `in`).

`__getitem__(index: int) → Attribute`

Доступ к элементу по индексу (`obj[i]`).

`__init__(*attributes: Attribute, coordsystem: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- ***attributes** - Атрибуты.
- **coordsystem** - Система координат для геометрического атрибута.

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__setitem__(index: int, value: Attribute)`

Установка элемента по индексу (`obj[i] = value`).

property attribute_names: List[str]

Возвращает список имен атрибутов.

by_name(n: str) → Optional[Attribute]

Возвращает атрибут по его имени. Если такого имени не существует, возвращает None.

Параметры

n - Наименование атрибута.

property coordsystem: Optional[CoordSystem]

Устанавливает или возвращает систему координат.

Результат

None, если СК отсутствует.

См.также:

`axipy.Table.is_spatial`

Список 48: 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
```

delete(index: Union[int, str])

Удаляет атрибут.

Параметры

index - Индекс удаляемого атрибута.

index_by_name(n: str) → int

Возвращает индекс по имени атрибута.

Параметры

n - Наименование атрибута.

insert(index: `int`, attr: `Attribute`)

Вставляє атрибут.

Параметри

- **index** – Індекс, по якому проводиться вставка.
- **attr** – Атрибут.

22.11 Attribute - Атрибут схеми таблиці

class `ахіру.Attribute`

Атрибут схеми таблиці.

Используется для создания и инспектирования атрибутов и схем `ахіру.Schema`. Для создания атрибутов используйте функции `string()`, `decimal()` и другие.

Список 49: Пример создания

```
attr = Attribute.string('attribute_name', 80)
attr.alias = 'Alias for attribute'
attr.readonly = True
attr.comments = 'Текстовое описание поля'
```

Конструктор класса:

<code>__init__</code> (name, typedef)	Конструктор класса.

Классовые методы:

<code>bool</code> (name)	Создает атрибут логического типа.
<code>date</code> (name)	Создает атрибут типа дата.
<code>datetime</code> (name)	Создает атрибут типа дата и время.
<code>decimal</code> (name[, length, precision])	Создает атрибут десятичного типа.
<code>double</code> (name)	Создает атрибут вещественного типа.
<code>float</code> (name)	Создает атрибут вещественного типа.
<code>integer</code> (name)	Создает атрибут целого типа.
<code>large</code> (name)	Создает атрибут целого 64-битного типа.
<code>short</code> (name)	Создает атрибут целого 16-битного типа.
<code>string</code> (name[, length])	Создает атрибут строкового типа.
<code>time</code> (name)	Создает атрибут типа время.

Свойства:

<code>alias</code>	Устанавливает или возвращает псевдоним.
<code>comments</code>	Устанавливает или возвращает дополнительную текстовую информацию к полю.
<code>length</code>	Возвращает длину атрибута.
<code>name</code>	Возвращает имя атрибута.
<code>precision</code>	Возвращает точность.
<code>readOnly</code>	Устанавливает или возвращает признак "Поле только для чтения".
<code>type_string</code>	Возвращает тип в виде строки без длины и точности.
<code>typedef</code>	Возвращает описание типа.
<code>unique</code>	Устанавливает или возвращает признак "Поле является уникальным".

Атрибуты:

<code>DEFAULT_DECIMAL_LENGTH</code>	Длина поля типа <code>decimal</code> по умолчанию.
<code>DEFAULT_DECIMAL_PRECISION</code>	Точность поля типа <code>decimal</code> по умолчанию.
<code>DEFAULT_STRING_LENGTH</code>	Длина строки по умолчанию.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`DEFAULT_DECIMAL_LENGTH: Final[int]`

Длина поля типа `decimal` по умолчанию.

`DEFAULT_DECIMAL_PRECISION: Final[int]`

Точность поля типа `decimal` по умолчанию.

`DEFAULT_STRING_LENGTH: Final[int]`

Длина строки по умолчанию.

`__eq__(other: object) → bool`

Оператор равенства (`==`).

`__init__(name: str, typedef: str)`

Конструктор класса.

Параметры

- **`name`** – Имя атрибута.
- **`typedef`** – Описание типа в формате `<тип>[ :длина] [ .точность]`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property alias: [Optional\[str\]](#)

Устанавливает или возвращает псевдоним.

static bool(name: [str](#)) → [Attribute](#)

Создает атрибут логического типа.

Параметры

name – Имя атрибута.

property comments: [Optional\[str\]](#)

Устанавливает или возвращает дополнительную текстовую информацию к полю.

static date(name: [str](#)) → [Attribute](#)

Создает атрибут типа дата.

Параметры

name – Имя атрибута.

static datetime(name: [str](#)) → [Attribute](#)

Создает атрибут типа дата и время.

Параметры

name – Имя атрибута.

static decimal(name: [str](#), length: [int](#) = DEFAULT_DECIMAL_LENGTH, precision: [int](#) = DEFAULT_DECIMAL_PRECISION) → [Attribute](#)

Создает атрибут десятичного типа.

Параметры

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

static double(name: [str](#)) → [Attribute](#)

Создает атрибут вещественного типа.

Параметры

name – Имя атрибута.

static float(name: [str](#)) → [Attribute](#)

Создает атрибут вещественного типа.

То же, что и [double\(\)](#)

Параметры

name – Имя атрибута.

static integer(name: [str](#)) → [Attribute](#)

Создает атрибут целого типа.

Параметры

name – Имя атрибута.

static large(name: [str](#)) → [Attribute](#)

Создает атрибут целого 64-битного типа.

Параметры

name – Имя атрибута.

property length: int

Возвращает длину атрибута.

property name: str

Возвращает имя атрибута.

property precision: int

Возвращает точность.

property readOnly: bool

Устанавливает или возвращает признак «Поле только для чтения».

static short(name: str) → [Attribute](#)

Создает атрибут целого 16-битного типа.

Параметры

name – Имя атрибута.

static string(name: str, length: int = DEFAULT_STRING_LENGTH) → [Attribute](#)

Создает атрибут строкового типа.

Параметры

- **name** – Имя атрибута.
- **length** – Длина атрибута.

static time(name: str) → [Attribute](#)

Создает атрибут типа время.

Параметры

name – Имя атрибута.

property type_string: str

Возвращает тип в виде строки без длины и точности.

property typedef: str

Возвращает описание типа.

Строка вида <тип>[:длина][.точность].

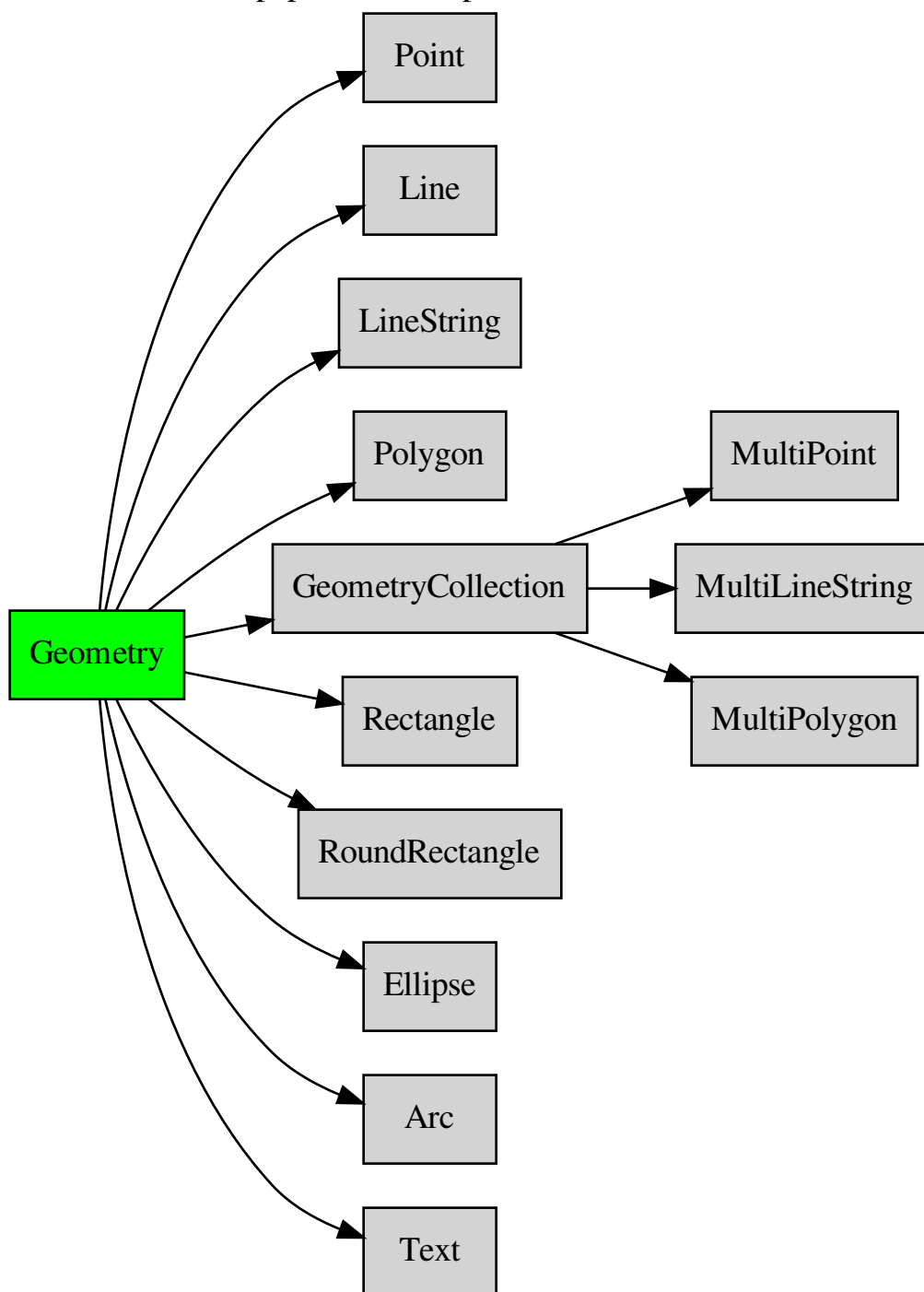
property unique: bool

Устанавливает или возвращает признак «Поле является уникальным».

22.12 Geometry - Геометрия

22.12.1 Geometry - Геометрия

Иерархия геометрических классов:



class axipy.Geometry

Абстрактный класс геометрического объекта (геометрии).

Примечание: Для получения краткого текстового представления геометрии можно воспользоваться функцией `str`. Для более полного - `repr()`.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии <code>distance</code> по направлению <code>azimuth</code> .

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.

continues on next page

Таблица 12 - продолжение с предыдущей страницы

<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.

continues on next page

Таблица 12 - продолжение с предыдущей страницы

<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`
Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans - Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 50: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 51: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional](#)[[CoordSystem](#)]

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], end: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], cs: [Optional](#)[[CoordSystem](#)] = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 53: Пример.

```

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)

```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 54: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    ↪ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ↪ ELLIPSOID))

...

>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem`.

supported_calc_modes.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 55: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 56: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsiod:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: `bool`

Проверяет геометрию на валидность.

property is_valid_reason: `str`

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: `str`

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.

- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.2 Point - Точечный объект

class axipy.[Point](#)

Базовые классы: [Geometry](#)

Геометрический объект типа точка.

Список 57: Пример.

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

Конструктор класса:

<code>__init__(x, y[, cs])</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.
<code>x</code>	X Координата.
<code>y</code>	Y Координата.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаямляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.

continues on next page

Таблица 13 - продолжение с предыдущей страницы

<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__init__(x: float, y: float, cs: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **x** – X координата.
- **y** – Y координата.
- **cs** – Система Координат, в которой создается геометрия.

`__repr__() → str`

Представление геометрии в виде строки.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 58: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```


Список 59: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional](#)[[CoordSystem](#)]

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], end: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], cs: [Optional](#)[[CoordSystem](#)] = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals (other: Geometry) $\rightarrow$ bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

```
static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry
```

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

```
static from_mif(mif: str) → Geometry
```

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

```
classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKB**.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 60: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

```
classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKT**.

Параметры

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

Список 61: Пример.

```

polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')

```

(continues on next page)

(продолжение с предыдущей страницы)

```
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 62: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 63: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 64: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsiod:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsiod: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: `Union[Pnt, Tuple[float, float]]`, azimuth: `float`, distance: `float`, cs: `Optional[CoordSystem]` = None) → `Pnt`

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↳ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↳ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

property x: float

X Координата.

property y: float

Y Координата.

22.12.3 Line - Линия

class axipy.Line

Базовые классы: [Geometry](#)

Геометрический объект типа линия.

Список 65: 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) # Заменяем координаты начальной точки.
line.end = Pnt(120, 120)
```

Конструктор класса:

<code>__init__(begin, end[, cs])</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>begin</code>	Начальная точка линии.
<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>end</code>	Конечная точка линии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.

continues on next page

Таблица 14 - продолжение с предыдущей страницы

<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`
Оператор равенства (==).

`__init__(begin: Pnt, end: Pnt, cs: Optional[CoordSystem] = None)`
Конструктор класса.

Параметры

- **begin** - Начальная точка линии.
- **end** - Конечная точка линии.
- **cs** - Система Координат, в которой создается геометрия.

`__repr__()` → `str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans - Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 66: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 67: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
```

(continues on next page)

(продолжение с предыдущей страницы)

```
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

property begin: [Pnt](#)

Начальная точка линии.

Точки допустимо создавать как экземпляры [Pnt](#) либо в виде пары float значений tuple

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центр оид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union\[Pnt, Tuple\[float, float\]\]](#), end: [Union\[Pnt, Tuple\[float, float\]\]](#), cs: [Optional\[CoordSystem\]](#) = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

property end: [Pnt](#)

Конечная точка линии.

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals(other: [Geometry](#)) → [bool](#)

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: [str](#), cs: [Optional\[CoordSystem\]](#) = None) → [Geometry](#)

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: [str](#)) → [Geometry](#)

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod from_wkb(wkb: [bytes](#), coordsystem: [Optional\[CoordSystem\]](#) = None) → [Geometry](#)

Создает геометрический объект из строки формата **WKB**.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 68: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

classmethod from_wkt(wkt: [str](#), coordsystem: [Optional\[CoordSystem\]](#) = None) → [Geometry](#)

Создает геометрический объект из строки формата **WKT**.

Параметры

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

Список 69: Пример.

```
polygon = Geometry.from_wkt('POLYGON ((10 10, 100 100, 100 10, 10 10))')
point = Geometry.from_wkt('SRID=4326;POINT (10 10)')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
crs = CoordSystem.from_epsg(4326)
pline = Geometry.from_wkt('LINESTRING (30 10, 10 30, 40 40)', crs)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 70: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 71: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 72: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: `Union[Pnt, Tuple[float, float]]`, azimuth: `float`, distance: `float`, cs: `Optional[CoordSystem]` = None) → `Pnt`

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↳ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↳ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.4 LineString - Полилиния

class ахіру.LineString

Базовые классы: `Geometry`

Геометрический объект типа полилиния.

Список 73: Пример.

```
csLL = CoordSystem.from_prj("1, 104")
ls = LineString([(1, 2), Pnt(3, 4), Pnt(5, 6), (7, 8)]) # Создадим полилинию без
↳СК
"""
Обновим точку с индексом 1. Допустимо только обновление точки целиком.
Изменение координат по одиночке не поддерживается.
"""
ls.points[1] = (33, 44)
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=csLL) # Создание полинии,
↳передав перечень точек.
itr = (a for a in ls.points) # Создадим итератор на базе точек первой полилинии
ls3 = LineString(itr)
```

Конструктор класса:

<code>__init__()</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>points</code>	Точки полилинии.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.

continues on next page

Таблица 15 - продолжение с предыдущей страницы

<code>get_line_direction()</code>	Направление линии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>set_line_direction(direction)</code>	Задание направления линии.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

```
__init__(points: Union[Iterable[Pnt], Iterable[Tuple[float, float]]], /, *, cs:
Optional[CoordSystem] = None) → None
```

```
__init__(*points: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None)
→ None
```

Конструктор класса.

Параметры

- **points** – Точки. Могут задаваться следующим образом:
 - В виде списка состоящего из объектов типа `axipy.Pnt` или `Tuple[float, float]`.
 - В виде перечня точек(`axipy.Pnt` или `Tuple[float, float]`) через запятую.
- **cs** – Система Координат, в которой создается геометрия. При задании, необходимо явно указать наименование параметра, `cs=`.

```
__len__() → int
```

Длина объекта (функция `len()`).

```
__repr__() → str
```

Представление геометрии в виде строки.

```
__str__() → str
```

Неформальное строковое представление объекта при вызове `str` или `print()`.

```
affine_transform(trans: QTransform) → Geometry
```

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 74: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 75: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional](#)[[CoordSystem](#)]

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], end: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], cs: [Optional](#)[[CoordSystem](#)] = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 77: Пример.

```

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)

```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 78: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem`.

supported_calc_modes.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 79: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 80: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsiod:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_line_direction() → `LineDirection`

Направление линии.

get_perimeter(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: Geometry) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: Geometry) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], azimuth: float, distance: float, cs: [Optional](#)[[CoordSystem](#)] = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property points: [ListPoint](#)

Точки полилинии. Реализован как список python [list](#) точек [Pnt](#). Также поддерживаются список пар [tuple](#).

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

relate(other: [Geometry](#)) → str

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], angle: float) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

set_line_direction(direction: [LineDirection](#))

Задание направления линии.

Параметры

direction – Желаемое значение

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```


to_polygon() → `Optional[Geometry]`

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → `bytes`

Возвращает WKB строку для геометрии.

to_wkt() → `str`

Возвращает WKT строку для геометрии.

touches(other: Geometry) → `bool`

Возвращает True, если геометрии соприкасаются.

property type: GeometryType

Возвращает тип геометрического элемента.

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

union(other: Geometry) → `Geometry`

Возвращает результат объединения двух геометрий.

within(other: Geometry) → `bool`

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.5 Polygon - Полигон

class axipy.Polygon

Базовые классы: `Geometry`

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

Список 81: 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
```

Конструктор класса:

`__init__()`

Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_rect(rect[, cs])</code>	Создает полигон на базе прямоугольника.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, azimuth, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>holes</code>	Дырки полигона.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>points</code>	Точки полигона.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

continues on next page

Таблица 16 - продолжение с предыдущей страницы

<code>convex_hull()</code>	Возвращает минимальный окаямляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_line_direction()</code>	Направление линии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>set_line_direction(direction)</code>	Задание направления линии.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.

continues on next page

Таблица 16 - продолжение с предыдущей страницы

<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(points: Union[Iterable[Pnt], Iterable[Tuple[float, float]]], /, *, cs: Optional[CoordSystem] = None) → None`

`__init__(*points: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → None`

Конструктор класса.

Параметры

- **points** - Точки. Могут задаваться следующим образом:
 - В виде списка состоящего из объектов типа `axipy.Pnt` или `Tuple[float, float]`.
 - В виде перечня точек(`axipy.Pnt` или `Tuple[float, float]`) через запятую.
- **cs** - Система Координат, в которой создается геометрия. При задании, необходимо явно указать наименование параметра, cs=.

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Представление геометрии в виде строки.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 82: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 83: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union\[Pnt, Tuple\[float, float\]\]](#), end: [Union\[Pnt, Tuple\[float, float\]\]](#), cs: [Optional\[CoordSystem\]](#) = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals(other: [Geometry](#)) → [bool](#)

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: [str](#), cs: [Optional\[CoordSystem\]](#) = None) → [Geometry](#)

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: [str](#)) → [Geometry](#)

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

static from_rect(rect: [Rect](#), cs: [Optional\[CoordSystem\]](#) = None) → [Polygon](#)

Создает полигон на базе прямоугольника.

Параметры

- **rect** - Прямоугольник, на основе которого формируются координаты.
- **cs** - Система Координат, в которой создается геометрия.

```
classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKB**.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 84: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

```
classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKT**.

Параметры

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

Список 85: Пример.

```

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)

```

```
get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] =
None) → float
```

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc mode в квадратных метрах.

Список 86: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
```

(continues on next page)

(продолжение с предыдущей страницы)

```

-> PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
-> ELLIPSOID))

...

>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 87: 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")

```

(continues on next page)

(продолжение с предыдущей страницы)

```
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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 88: 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('Length Mercator m:', ls.get_length(Unit.m))
print('Length Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Length LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
    ↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Length Mercator m: 9988.805508567782
>>> Length Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Length LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_line_direction() → LineDirection

Направление линии.

get_perimeter(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

`ValueError`, `RuntimeError`, `RuntimeWarning` –

property holes: `ListHoles`

Дырки полигона. Реализован в виде списка списков точек.

Список 89: Пример.

```
poly = Polygon((0, 0), (1, 10), (10, 1))
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
# Изменим направление для полигона
poly.set_line_direction(LineDirection.CounterClockwise)
# То же, но для дырки
poly.holes[0].set_line_direction(LineDirection.CounterClockwise)
```

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: `bool`

Проверяет геометрию на валидность.

property is_valid_reason: `str`

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: `str`

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: `Union[Pnt, Tuple[float, float]]`, azimuth: `float`, distance: `float`, cs: `Optional[CoordSystem]` = None) → `Pnt`

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property points: `ListPoint`

Точки полигона.

Реализован как список python `list` точек `Pnt`. В каждом контуре, количество точек на 1 больше, чем количество узлов, так как контур - это замкнутая линия, и первая и последняя точка совпадают.

relate(other: `Geometry`) → `str`

Проверяет отношения между объектами.

Подробнее см. `DE-9IM`

reproject(cs: `CoordSystem`) → `Geometry`

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: `Union[Pnt, Tuple[float, float]]`, angle: `float`) → `Geometry`

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: `float`, ky: `float`) → `Geometry`

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

set_line_direction(direction: `LineDirection`)

Задание направления линии.

Параметры

direction – Желаемое значение

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → bool

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.6 ListPoint - Список точек

class axipy.[ListPoint](#)

Методы:

append (p)	Добавляет точку в конец.
get_line_direction ()	Возвращает направление точек.
insert (idx, p)	Вставляет точку по индексу.
pop (idx)	Удаляет и возвращает точку.
remove (idx)	Удаляет точку по индексу.
set_line_direction (direction)	Устанавливает направление точек.

Специальные методы:

__contains__ (item)	Проверяет наличие элемента в объекте (оператор in).
__getitem__ (idx)	Доступ к элементу по индексу (obj[i]).
__iter__ ()	Возвращает итератор по объекту (функция iter ()).
__len__ ()	Длина объекта (функция len ()).
__setitem__ (idx, p)	Установка элемента по индексу (obj[i] = value).
__str__ ()	Неформальное строковое представление объекта при вызове str или print ().

[__contains__](#)(item: [Pnt](#)) → bool

Проверяет наличие элемента в объекте (оператор in).

[__getitem__](#)(idx: int) → [Pnt](#)

Доступ к элементу по индексу (obj[i]).

[__iter__](#)() → [Iterator](#)[[Pnt](#)]

Возвращает итератор по объекту (функция [iter](#)()).

`__len__()` → `int`
 Длина объекта (функция `len()`).

`__setitem__(idx: int, p: Union[Pnt, Tuple[float, float]])`
 Установка элемента по индексу (`obj[i] = value`).

`__str__()` → `str`
 Неформальное строковое представление объекта при вызове `str` или `print()`.

`append(p: Union[Pnt, Tuple[float, float]])`
 Добавляет точку в конец.

`get_line_direction()` → `LineDirection`
 Возвращает направление точек.

`insert(idx: int, p: Union[Pnt, Tuple[float, float]])`
 Вставляет точку по индексу.

`pop(idx: int)` → `Pnt`
 Удаляет и возвращает точку.

`remove(idx: int)`
 Удаляет точку по индексу.

`set_line_direction(direction: LineDirection)`
 Устанавливает направление точек.

22.12.7 ListHoles - Список дырок

`class axipy.ListHoles`

Методы:

<code>append(*points)</code>	Добавляет точки как новую дырку.
<code>remove(idx)</code>	Удаляет дырку по индексу.

Специальные методы:

<code>__contains__(item)</code>	Проверяет наличие элемента в объекте (оператор <code>in</code>).
<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__contains__(item: ListPoint)` → `bool`
 Проверяет наличие элемента в объекте (оператор `in`).

`__getitem__(idx: int)` → `ListPoint`
 Доступ к элементу по индексу (`obj[i]`).

`__iter__()` → `Iterator[ListPoint]`
 Возвращает итератор по объекту (функция `iter()`).

`__len__()` → `int`

Длина объекта (функция `len()`).

`append(*points: Union[Pnt, Tuple[float, float], Iterable[Pnt], Iterable[Tuple[float, float]]])`

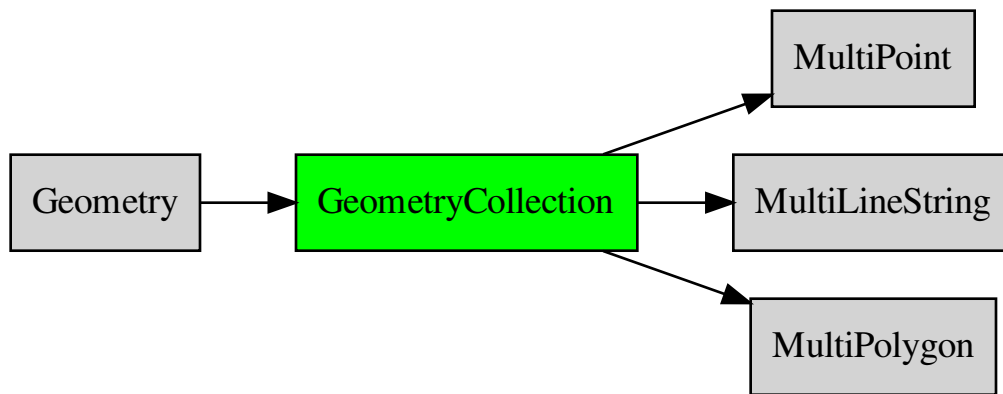
Добавляет точки как новую дырку.

`remove(idx: int)`

Удаляет дырку по индексу.

22.12.8 GeometryCollection - Коллекция геометрий

Иерархия геометрических классов:



class `axipy.GeometryCollection`

Базовые классы: `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 – 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. Также допустимо изменение некоторых свойств геометрии в зависимости от ее типа.


```
coll.append((3, 4), (5, 5), (10, 0)) # Полилиния
coll.append(Polygon([(3, 4), (5, 5), (10, 0)])) # Полигон

# 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

# Доступ к элементам коллекции::
for g in coll:
    print('Геометрия:', g)
```

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>append(value)</code>	Добавление геометрии в коллекцию.

continues on next page

Таблица 17 - продолжение с предыдущей страницы

<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>remove(idx)</code>	Удаление геометрии из коллекции.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.

continues on next page

Таблица 17 - продолжение с предыдущей страницы

<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>try_to_simplified()</code>	Создает копию коллекции при этом пробует упростить ее тип.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__contains__(item)</code>	Проверяет наличие элемента в объекте (оператор in).
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__setitem__(idx, value)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__contains__(item: Union[Point, LineString, Polygon]) → bool`

Проверяет наличие элемента в объекте (оператор in).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__getitem__(idx: int) → Union[Point, LineString, Polygon]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[Union[Point, LineString, Polygon]]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__()` → `str`

Представление геометрии в виде строки.

`__setitem__(idx: int, value: Union[Point, LineString, Polygon])`

Установка элемента по индексу (`obj[i] = value`).

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 90: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 91: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: Geometry, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

append(value: Union[Point, LineString, Polygon])

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**.

Список 92: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
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`.
```

boundary() → Optional[Geometry]

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional](#)[[CoordSystem](#)]

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], end: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], cs: [Optional](#)[[CoordSystem](#)] = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 94: Пример.

```

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)

```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 95: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem`.

supported_calc_modes.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 96: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 97: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsiod:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Length LL km: 1111.9487428468117
>>> Length LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: `bool`

Проверяет геометрию на валидность.

property is_valid_reason: `str`

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: `str`

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

remove(idx: [int](#))

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↳ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↳ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...

```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → Union[[MultiPoint](#), [MultiLineString](#), [MultiPolygon](#), [GeometryCollection](#)]

Создает копию коллекции при этом пробует упростить ее тип.

К примеру, если в коллекции только полигоны, то вернет объект типа [MultiPolygon](#). Если исходная коллекция разнородна, возвращается копия исходной.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.9 MultiPoint - Коллекция точек

class axipy.[MultiPoint](#)

Базовые классы: [GeometryCollection](#)

Коллекция точечных объектов. Может содержать только объекты типа точка.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 98: 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 # Заменяем только координату x для первого объекта в коллекции
mpoint.remove(1) # Удаляем второй объект
```

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, azimuth, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.

continues on next page

Таблица 18 - продолжение с предыдущей страницы

<code>append(value)</code>	Добавление геометрии в коллекцию.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>remove(idx)</code>	Удаление геометрии из коллекции.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.

continues on next page

Таблица 18 - продолжение с предыдущей страницы

<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>try_to_simplified()</code>	Создает копию коллекции при этом пробует упростить ее тип.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__contains__(item)</code>	Проверяет наличие элемента в объекте (оператор in).
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__setitem__(idx, value)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__contains__(item: Union[Point, LineString, Polygon]) → bool`

Проверяет наличие элемента в объекте (оператор in).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__getitem__(idx: int) → Union[Point, LineString, Polygon]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[Point]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__()` → `str`

Представление геометрии в виде строки.

`__setitem__(idx: int, value: Union[Point, LineString, Polygon])`

Установка элемента по индексу (`obj[i] = value`).

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 99: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 100: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

append(value: [Union](#)[[Point](#), [LineString](#), [Polygon](#)])

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](#).

Список 101: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
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`.
```

boundary() → [Optional](#)[[Geometry](#)]

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** - Ширина буфера.
- **resolution** - Количество сегментов на квадрант.
- **capStyle** - Стил ь окончания.
- **join_style** - Стил ь соединения.
- **mitreLimit** - Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union\[Pnt, Tuple\[float, float\]\]](#), end: [Union\[Pnt, Tuple\[float, float\]\]](#), cs: [Optional\[CoordSystem\]](#) = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** - Начальная точка. Если задана СК, то координаты в ней.
- **end** - Конечная точка. Если задана СК, то координаты в ней.
- **cs** - СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

- **coordsystem** – Система координат, которая будет установлена для геометрии. Если строка задана в виде EWKT и указано значение SRID, игнорируется.

Список 103: Пример.

```

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)

```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 104: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    ↪ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ↪ ELLIPSOID))

...

>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem`.

supported_calc_modes.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_distance(other: Geometry, u: Optional[LinearUnit] = None) → float

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 105: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 106: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsiod:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: `bool`

Проверяет геометрию на валидность.

property is_valid_reason: `str`

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: `str`

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

remove(idx: [int](#))

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → Optional[Geometry]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → bytes

Возвращает WKB строку для геометрии.

to_wkt() → str

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → bool

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → [Union](#)[[MultiPoint](#), [MultiLineString](#), [MultiPolygon](#), [GeometryCollection](#)]

Создает копию коллекции при этом пробует упростить ее тип.

К примеру, если в коллекции только полигоны, то вернет объект типа [MultiPolygon](#). Если исходная коллекция разнородна, возвращается копия исходной.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → bool

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.10 MultiLineString - Коллекция полилиний

class ахіру.[MultiLineString](#)

Базовые классы: [GeometryCollection](#)

Коллекция полилиний. Может содержать только объекты типа полилиния.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 107: Пример.

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

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>append(value)</code>	Добавление геометрии в коллекцию.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

continues on next page

Таблица 19 - продолжение с предыдущей страницы

<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>remove(idx)</code>	Удаление геометрии из коллекции.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.

continues on next page

Таблица 19 - продолжение с предыдущей страницы

<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>try_to_simplified()</code>	Создает копию коллекции при этом пробует упростить ее тип.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__contains__(item)</code>	Проверяет наличие элемента в объекте (оператор in).
<code>__eq__(other)</code>	Оператор равенства (==).
<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__setitem__(idx, value)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__contains__(item: Union[Point, LineString, Polygon]) → bool`

Проверяет наличие элемента в объекте (оператор in).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__getitem__(idx: int) → Union[Point, LineString, Polygon]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[LineString]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Представление геометрии в виде строки.

`__setitem__(idx: int, value: Union[Point, LineString, Polygon])`

Установка элемента по индексу (`obj[i] = value`).

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 108: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 109: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

append(value: [Union](#)[[Point](#), [LineString](#), [Polygon](#)])

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](#).

Список 110: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
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`.
```

boundary() → [Optional](#)[[Geometry](#)]

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional](#)[[CoordSystem](#)]

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → bool

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], end: [Union](#)[[Pnt](#), [Tuple](#)[float, float]], cs: [Optional](#)[[CoordSystem](#)] = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals(other: [Geometry](#)) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.


```
static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry
```

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

```
static from_mif(mif: str) → Geometry
```

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

```
classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата [WKB](#).

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 111: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

```
classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKT**.

Параметры

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

Список 112: Пример.

```

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)

```

```
get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] =
None) → float
```

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc mode в квадратных метрах.

Список 113: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_distance(other: `Geometry`, u: `Optional[LinearUnit]` = None) → float

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 114: 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None)
→ float

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

Список 115: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: [Optional\[LinearUnit\]](#) = None, calc_mode: [Optional\[CalcMode\]](#) = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. [get_area\(\)](#)

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии [CoordSystem.supported_calc_modes](#).

Исключение

ValueError, RuntimeError, RuntimeWarning -

intersection(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область пересечения с другой геометрией.

intersects(other: [Geometry](#)) → bool

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → bool

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: float, distance: float, cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** - Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** - Азимут в градусах, указывающий направление.
- **distance** - Расстояние по азимуту. Задается в метрах.

- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

remove(idx: [int](#))

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↳((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↳5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → [Union\[MultiPoint, MultiLineString, MultiPolygon, GeometryCollection\]](#)

Создает копию коллекции при этом пробует упростить ее тип.

К примеру, если в коллекции только полигоны, то вернет объект типа [MultiPolygon](#). Если исходная коллекция разнородна, возвращается копия исходной.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.11 MultiPolygon - Коллекция полигонов

class axipy.**MultiPolygon**

Базовые классы: [GeometryCollection](#)

Коллекция полигонов. Может содержать только объекты типа полигон.

Параметры

cs – Система Координат, в которой создается геометрия.

Список 116: Пример.

```
poly = Polygon((0, 0), (1, 10), (10, 1))
poly.holes.append([(2, 2), (2, 4), (5, 3)]) # Добавим дырку
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
mpoly[1].holes[0][1] = (99, 99) # Изменение второй точки дырки
mpoly[1].points[0] = (0, 0) # Заменяем первую (она же последняя) точку полигона
poly2 = Polygon([(11, 12), (13, 14), (15, 16), (17, 18)])
mpoly[0] = poly2 # Полностью заменим первый полигон
```

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>append(value)</code>	Добавление геометрии в коллекцию.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.

continues on next page

Таблица 20 - продолжение с предыдущей страницы

<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>remove(idx)</code>	Удаление геометрии из коллекции.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>try_to_simplified()</code>	Создает копию коллекции при этом пробует упростить ее тип.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__contains__(item)</code>	Проверяет наличие элемента в объекте (оператор <code>in</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.
<code>__setitem__(idx, value)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__contains__(item: Union[Point, LineString, Polygon]) → bool`

Проверяет наличие элемента в объекте (оператор `in`).

`__eq__(other: object) → bool`

Оператор равенства (`==`).

`__getitem__(idx: int) → Union[Point, LineString, Polygon]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[Polygon]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Представление геометрии в виде строки.

`__setitem__(idx: int, value: Union[Point, LineString, Polygon])`

Установка элемента по индексу (`obj[i] = value`).

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 117: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 118: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: *Geometry*, tolerance: *float*) → *bool*

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

append(value: Union[Point, LineString, Polygon])

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.

Список 119: Пример.

```
# Создадим коллекцию и добавим в нее несколько объектов различного типа.
coll = GeometryCollection()
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`.
```

boundary() → Optional[Geometry]

Возвращает границы геометрии в виде полилинии.

property bounds: Rect

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: float, resolution: int = 16, capStyle: LineCapStyle = LineCapStyle.Round, join_style: LineJoinStyle = LineJoinStyle.Round, mitreLimit: float = 5.0) → Geometry

Производит построение буфера вокруг объекта.

Параметры

- **distance** - Ширина буфера.
- **resolution** - Количество сегментов на квадрант.
- **capStyle** - Стил ь окончания.
- **join_style** - Стил ь соединения.
- **mitreLimit** - Предел среза.

centroid() → Point

Возвращает центроид геометрии.

clone() → Geometry

Создает копию объекта.

contains(other: Geometry) → bool

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → Geometry

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: `Optional[CoordSystem]`

Система Координат (СК) геометрии.

covers(other: `Geometry`) → `bool`

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: `Geometry`) → `bool`

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: `Geometry`) → `Optional[Geometry]`

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: `Geometry`) → `bool`

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: `Union[Pnt, Tuple[float, float]]`, end: `Union[Pnt, Tuple[float, float]]`, cs: `Optional[CoordSystem] = None`) → `Tuple`

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → `Geometry`

Возвращает полигон, описывающий заданную геометрию.

equals(other: `Geometry`) → `bool`

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: `str`, cs: `Optional[CoordSystem] = None`) → `Geometry`

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static `from_mif(mif: str) → Geometry`

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod `from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry`

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 120: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

classmethod `from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry`

Создает геометрический объект из строки формата WKT.

Параметры

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

Список 121: Пример.

```
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)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 122: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    ↪ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ↪ ELLIPSOID))

...

>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 123: 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")

```

(continues on next page)

(продолжение с предыдущей страницы)

```
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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 124: 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('Length Mercator m:', ls.get_length(Unit.m))
print('Length Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Length LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
'''

>>> Length Mercator km: 9.988805508567783
>>> Length Mercator m: 9988.805508567782
>>> Length Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Length LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_perimeter(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных

объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: `Union[Pnt, Tuple[float, float]]`, azimuth: `float`, distance: `float`, cs: `Optional[CoordSystem]` = None) → `Pnt`

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

remove(idx: [int](#))

Удаление геометрии из коллекции.

Параметры

idx – Индекс геометрии в коллекции.

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional\[Geometry\]](#)

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↳ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↳ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...

```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional](#)[[Geometry](#)]

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

try_to_simplified() → [Union](#)[[MultiPoint](#), [MultiLineString](#), [MultiPolygon](#), [GeometryCollection](#)]

Создает копию коллекции при этом пробует упростить ее тип.

К примеру, если в коллекции только полигоны, то вернет объект типа [MultiPolygon](#). Если исходная коллекция разнородна, возвращается копия исходной.

property type: GeometryType

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

`within(other: Geometry) → bool`

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.12 Rectangle - Прямоугольник

class axipy.Rectangle

Базовые классы: [Geometry](#)

Геометрический объект типа прямоугольник.

Список 125: 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`.
```

Конструктор класса:

<code>__init__(*par[, cs])</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.
<code>xmax</code>	Максимальное значение X.
<code>xmin</code>	Минимальное значение X.
<code>ymax</code>	Максимальное значение Y.
<code>ymin</code>	Минимальное значение Y.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаямляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.

continues on next page

Таблица 21 - продолжение с предыдущей страницы

<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта <code>other</code> .
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает <code>True</code> , если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает <code>True</code> , если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает <code>True</code> , если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает <code>True</code> , если геометрия находится полностью внутри геометрии <code>other</code> .

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__init__(*par: Union[Rect, float], cs: Optional[CoordSystem] = None)`
Конструктор класса.

Параметры

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

`__repr__()` → `str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 126: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 127: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стиль окончания.
- **join_style** – Стиль соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → *Geometry*

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: *Optional[CoordSystem]*

Система Координат (СК) геометрии.

covers(other: *Geometry*) → *bool*

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: *Geometry*) → *bool*

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: *Geometry*) → *Optional[Geometry]*

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: *Geometry*) → *bool*

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: *Union[Pnt, Tuple[float, float]]*, end: *Union[Pnt, Tuple[float, float]]*, cs: *Optional[CoordSystem]* = None) → *Tuple*

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция *point_by_azimuth()*

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → *Geometry*

Возвращает полигон, описывающий заданную геометрию.

equals(other: *Geometry*) → *bool*

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: *str*, cs: *Optional[CoordSystem]* = None) → *Geometry*

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.

- **cs** – Система координат.

```
static from mif(mif: str) → Geometry
```

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

```
classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKB**.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 128: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

```
classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry
```

Создает геометрический объект из строки формата **WKT**.

Параметры

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

Список 129: Пример.

```

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)

```

```
get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] =
None) → float
```

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc mode в квадратных метрах.

Список 130: Пример.

```
# В проекции Меркатора:
csMercatorKm = CoordSystem.from_prj("10, 104, 1, 0")
poly = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)], cs=csMercatorKm)
```

(continues on next page)

(продолжение с предыдущей страницы)

```

print('Area Mercator sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
...

```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 131: Пример.

```

# Создадим геометрию без СК
first = Point(0, 0)
second = Point(10, 0)
print('В плане:', first.get_distance(second))

```

(continues on next page)

(продолжение с предыдущей страницы)

```
# 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
'''
```

get_length(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

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

Список 132: 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('Length Mercator m:', ls.get_length(Unit.m))
print('Length Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Length LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
    ELLIPSOID))
'''
>>> Length Mercator km: 9.988805508567783
>>> Length Mercator m: 9988.805508567782
>>> Length Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Length LL km on ellipsoid: 1113.1949079327358
'''
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: Optional[LinearUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

intersection(other: `Geometry`) → `Optional[Geometry]`

Возвращает область пересечения с другой геометрией.

intersects(other: `Geometry`) → `bool`

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: `Geometry`) → `bool`

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: `Union[Pnt, Tuple[float, float]]`, azimuth: `float`, distance: `float`, cs: `Optional[CoordSystem]` = None) → `Pnt`

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция `distance_by_points()`

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: [float](#), ky: [float](#)) → [Geometry](#)

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: [float](#), dy: [float](#)) → [Geometry](#)

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪ ((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪ 5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...
```

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

property xmax: [float](#)

Максимальное значение X.

property xmin: [float](#)

Минимальное значение X.

property ymax: [float](#)

Максимальное значение Y.

property ymin: [float](#)

Минимальное значение Y.

22.12.13 RoundedRectangle - Скругленный прямоугольник

class ахіру.RoundedRectangle

Базовые классы: [Rectangle](#)

Геометрический объект типа скругленный прямоугольник.

Список 133: Пример.

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

Конструктор класса:

<code>__init__(rect, xRad, yRad[, cs])</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.
<code>xRadius</code>	Скругление углов по координате X.
<code>xmax</code>	Максимальное значение X.
<code>xmin</code>	Минимальное значение X.
<code>yRadius</code>	Скругление углов по координате Y.
<code>ymax</code>	Максимальное значение Y.
<code>ymin</code>	Минимальное значение Y.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.

continues on next page

Таблица 22 - продолжение с предыдущей страницы

<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(rect: Union[Rect, list], xRad: float, yRad: float, cs: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **rect** – Прямоугольник класса `Rect` или как `list`.
- **xRad** – Скругление по X.
- **yRad** – Скругление по Y.
- **cs** – Система Координат, в которой создается геометрия.

`__repr__() → str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 134: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 135: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

`almost_equals(other: Geometry, tolerance: float) → bool`

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals(other: [Geometry](#)) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → [Geometry](#)

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → [Geometry](#)

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → [Geometry](#)

Создает геометрический объект из строки формата [WKB](#).

Параметры

- **wkb** – Строка WKB

- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 136: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

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

Список 137: Пример.

```
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)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 138: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ELLIPSOID))
```

(continues on next page)

(продолжение с предыдущей страницы)

```

'''
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
'''

```

Параметры

- **area_unit** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение**ValueError, RuntimeError, RuntimeWarning -****get_distance**(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 139: 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
'''

```

get_length(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает длину геометрии. Метод применим только для линейных

объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 140: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
→ ELLIPSOID))
...
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
...
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

intersection(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область пересечения с другой геометрией.

intersects(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → [bool](#)

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → str

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → `Optional[Geometry]`

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → `bytes`

Возвращает WKB строку для геометрии.

to_wkt() → `str`

Возвращает WKT строку для геометрии.

touches(other: Geometry) → `bool`

Возвращает True, если геометрии соприкасаются.

property type: GeometryType

Возвращает тип геометрического элемента.

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

union(other: Geometry) → `Geometry`

Возвращает результат объединения двух геометрий.

within(other: Geometry) → `bool`

Возвращает True, если геометрия находится полностью внутри геометрии other.

property xRadius: float

Скругление углов по координате X.

property xmax: float

Максимальное значение X.

property xmin: float

Минимальное значение X.

property yRadius: float

Скругление углов по координате Y.

property ymax: float

Максимальное значение Y.

property ymin: float

Минимальное значение Y.

22.12.14 Ellipse - Эллипс

class ахіру.Ellipse

Базовые классы: `Geometry`

Геометрический объект типа эллипс.

Список 141: 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
```

Конструктор класса:

<code>__init__(rect[, cs])</code>	Конструктор класса.

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>center</code>	Центр эллипса.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>majorSemiAxis</code>	Радиус большой полуоси эллипса.
<code>minorSemiAxis</code>	Радиус малой полуоси эллипса.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>type</code>	Возвращает тип геометрического элемента.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.
<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.

continues on next page

Таблица 23 - продолжение с предыдущей страницы

<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(rect: Union[Rect, list], cs: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **rect** - Прямоугольник класса `Rect` или как `list`.
- **cs** - Система Координат, в которой создается геометрия.

`__repr__() → str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans - Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 142: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 143: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

`almost_equals(other: Geometry, tolerance: float) → bool`

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** - Сравниваемый объект.
- **tolerance** - Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** - Ширина буфера.
- **resolution** - Количество сегментов на квадрант.
- **capStyle** - Стил ь окончания.
- **join_style** - Стил ь соединения.
- **mitreLimit** - Предел среза.

property center: [Pnt](#)

Центр эллипса.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция [point_by_azimuth\(\)](#)

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

envelope() → [Geometry](#)

Возвращает полигон, описывающий заданную геометрию.

equals(other: [Geometry](#)) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → [Geometry](#)

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → [Geometry](#)

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → [Geometry](#)

Создает геометрический объект из строки формата [WKB](#).

Параметры

- **wkb** – Строка WKB

- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 144: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

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

Список 145: Пример.

```
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)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 146: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ELLIPSOID))
```

(continues on next page)

(продолжение с предыдущей страницы)

```

'''
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
'''

```

Параметры

- **area_unit** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение**ValueError, RuntimeError, RuntimeWarning -****get_distance**(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 147: 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
'''

```

get_length(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает длину геометрии. Метод применим только для линейных

объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 148: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
→ ELLIPSOID))
...
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
...
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

intersection(other: [Geometry](#)) → [Optional](#)[[Geometry](#)]

Возвращает область пересечения с другой геометрией.

intersects(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии пересекаются.

property is_valid: [bool](#)

Проверяет геометрию на валидность.

property is_valid_reason: [str](#)

Если геометрия неправильная, возвращает краткую аннотацию причины.

property majorSemiAxis: [float](#)

Радиус большой полуоси эллипса.

property minorSemiAxis: [float](#)

Радиус малой полуоси эллипса.

property name: [str](#)

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → [bool](#)

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union](#)[[Pnt](#), [Tuple](#)[[float](#), [float](#)]], azimuth: [float](#), distance: [float](#), cs: [Optional](#)[[CoordSystem](#)] = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...
```

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → `Optional[Geometry]`

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → `str`

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → `Optional[Geometry]`

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → `bytes`

Возвращает WKB строку для геометрии.

to_wkt() → `str`

Возвращает WKT строку для геометрии.

touches(other: Geometry) → `bool`

Возвращает True, если геометрии соприкасаются.

property type: GeometryType

Возвращает тип геометрического элемента.

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

union(other: Geometry) → `Geometry`

Возвращает результат объединения двух геометрий.

within(other: Geometry) → `bool`

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.15 Arc - Дуга

class axipy.Arc

Базовые классы: `Geometry`

Геометрический объект типа дуга.

Список 149: Пример.

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

Конструктор класса:

```
__init__(rect, startAngle, endAngle[, Конструктор класса.
cs])
```

Классовые методы:

<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, distance[, cs])</code>	azimuth, Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

<code>bounds</code>	Возвращает минимальный ограничивающий прямоугольник.
<code>center</code>	Центр дуги.
<code>coordsystem</code>	Система Координат (СК) геометрии.
<code>endAngle</code>	Конечный угол дуги.
<code>is_valid</code>	Проверяет геометрию на валидность.
<code>is_valid_reason</code>	Если геометрия неправильная, возвращает краткую аннотацию причины.
<code>name</code>	Возвращает наименование геометрического объекта.
<code>startAngle</code>	Начальный угол дуги.
<code>type</code>	Возвращает тип геометрического элемента.
<code>xRadius</code>	Радиус большой полуоси прямоугольника, в который вписана дуга.
<code>yRadius</code>	Радиус малой полуоси прямоугольника, в который вписана дуга.

Методы:

<code>affine_transform(trans)</code>	Трансформирует объект исходя из заданных параметров трансформации.
<code>almost_equals(other, tolerance)</code>	Производит примерное сравнения с другой геометрией в пределах заданной точности.
<code>boundary()</code>	Возвращает границы геометрии в виде полилинии.
<code>buffer(distance[, resolution, capStyle, ...])</code>	Производит построение буфера вокруг объекта.
<code>centroid()</code>	Возвращает центроид геометрии.
<code>clone()</code>	Создает копию объекта.

continues on next page

Таблица 24 - продолжение с предыдущей страницы

<code>contains(other)</code>	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
<code>convex_hull()</code>	Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.
<code>covers(other)</code>	Возвращает True, если геометрия охватывает геометрию other.
<code>crosses(other)</code>	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
<code>difference(other)</code>	Возвращает область первой геометрии, которая не пересечена второй геометрией.
<code>disjoint(other)</code>	Возвращает True, если геометрии не пересекаются и не соприкасаются.
<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.

continues on next page

Таблица 24 - продолжение с предыдущей страницы

<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(rect: Union[Rect, list], startAngle: float, endAngle: float, cs: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **rect** - Прямоугольник класса `Rect` или как `list`.
- **startAngle** - Начальный угол дуги.
- **endAngle** - Конечный угол дуги.
- **cs** - Система Координат, в которой создается геометрия.

`__repr__() → str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans - Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 150: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 151: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: *Geometry*, tolerance: float) → bool

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: [float](#), resolution: [int](#) = 16, capStyle: [LineCapStyle](#) = [LineCapStyle.Round](#), join_style: [LineJoinStyle](#) = [LineJoinStyle.Round](#), mitreLimit: [float](#) = 5.0) → [Geometry](#)

Производит построение буфера вокруг объекта.

Параметры

- **distance** – Ширина буфера.
- **resolution** – Количество сегментов на квадрант.
- **capStyle** – Стил ь окончания.
- **join_style** – Стил ь соединения.
- **mitreLimit** – Предел среза.

property center: [Pnt](#)

Центр дуги.

centroid() → [Point](#)

Возвращает центроид геометрии.

clone() → [Geometry](#)

Создает копию объекта.

contains(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → [Geometry](#)

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: [Optional\[CoordSystem\]](#)

Система Координат (СК) геометрии.

covers(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия охватывает геометрию other.

crosses(other: [Geometry](#)) → [bool](#)

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: Union[Pnt, Tuple[float, float]], end: Union[Pnt, Tuple[float, float]], cs: Optional[CoordSystem] = None) → Tuple

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

property endAngle: float

Конечный угол дуги.

envelope() → Geometry

Возвращает полигон, описывающий заданную геометрию.

equals(other: Geometry) → bool

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: str, cs: Optional[CoordSystem] = None) → Geometry

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: str) → Geometry

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod from_wkb(wkb: bytes, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKB.

Параметры

- **wkb** – Строка WKB

- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 152: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00'
pnt = Geometry.from_wkb(wkb)
```

classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

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

Список 153: Пример.

```
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)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 154: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
    PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
    ELLIPSOID))
```

(continues on next page)

(продолжение с предыдущей страницы)

```

'''
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
'''

```

Параметры

- **area_unit** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение**ValueError, RuntimeError, RuntimeWarning -****get_distance**(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** - Анализируемый объект.
- **u** - Единицы измерения, в которых требуется получить результат.

Список 155: 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
'''

```

get_length(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает длину геометрии. Метод применим только для линейных

объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Список 156: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
→ ELLIPSOID))
...
>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
...
```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

intersection(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область пересечения с другой геометрией.

intersects(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → [bool](#)

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: [Union\[Pnt, Tuple\[float, float\]\]](#), angle: [float](#)) → [Geometry](#)

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
'''
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
'''
```

property startAngle: float

Начальный угол дуги.

symmetric_difference(other: Geometry) → Geometry

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

to_geojson() → str

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → Optional[Geometry]

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → *str*

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → *Optional[Geometry]*

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → *bytes*

Возвращает WKB строку для геометрии.

to_wkt() → *str*

Возвращает WKT строку для геометрии.

touches(other: Geometry) → *bool*

Возвращает True, если геометрии соприкасаются.

property type: GeometryType

Возвращает тип геометрического элемента.

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

union(other: Geometry) → *Geometry*

Возвращает результат объединения двух геометрий.

within(other: Geometry) → *bool*

Возвращает True, если геометрия находится полностью внутри геометрии other.

property xRadius: float

Радиус большой полуоси прямоугольника, в который вписана дуга.

property yRadius: float

Радиус малой полуоси прямоугольника, в который вписана дуга.

22.12.16 Text - Текст

class axipy.Text

Базовые классы: *Geometry*

Геометрический объект типа текст.

Предупреждение: Геометрия текста и, в отличие от остальных типов объектов, определяется кроме геометрического представления так же и стилем его оформления *axipy.TextStyle*. Так же стоит заметить, что параметры геометрии текста зависит от того, куда (с карту или отчет) будет добавлен созданный текст. Поэтому созданный объект для карты должен быть добавлен в карту, а для отчета - в отчет. Подробнее см. *create_by_style()*.

Примечание: Для создания текстового объекта с указанием его размера в пунктах можно использовать метод `create_by_style()`.

Список 157: Пример создания текста и вставки его в отчет.

```
report_view = view_manager.create_reportview()
r = Rect(8, 6, 11, 7)
text = Text("Пример\ntекста", rect=r, angle=20, view=report_view)
geomItem = GeometryReportItem()
geomItem.geometry = text
geomItem.style = Style.for_geometry(text)
report_view.report.items.add(geomItem)
```

Конструктор класса:

<code>__init__(text, rect[, view, angle, cs])</code>	Конструктор класса.

Классовые методы:

<code>create_by_style(text, point, style, view[, ...])</code>	Создает текстовый объект по точке привязки и стиля для карты или отчета.
<code>distance_by_points(start, end[, cs])</code>	Производит расчет расстояния между двумя точками и азимут от первой до второй точки.
<code>from_geojson(json[, cs])</code>	Возвращает геометрию из ее 'GeoJSON' представления.
<code>from_mif(mif)</code>	Возвращает геометрию из ее 'MIF' представления.
<code>from_wkb(wkb[, coordsystem])</code>	Создает геометрический объект из строки формата WKB .
<code>from_wkt(wkt[, coordsystem])</code>	Создает геометрический объект из строки формата WKT .
<code>point_by_azimuth(point, azimuth, distance[, cs])</code>	Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

Свойства:

angle	Угол поворота текста, отсчитываемый от горизонтали против часовой стрелки.
bounds	Возвращает минимальный ограничивающий прямоугольник.
coordsystem	Система Координат (СК) геометрии.
endPoint	Координаты точки выноски (указки).
height	Высота прямоугольника с текстом.
is_valid	Проверяет геометрию на валидность.
is_valid_reason	Если геометрия неправильная, возвращает краткую аннотацию причины.
name	Возвращает наименование геометрического объекта.
rect_as_polygon	Представление ограничивающего прямоугольника в виде полигона.
startPoint	Координаты точки привязки.
text	Текст.
type	Возвращает тип геометрического элемента.
width	Ширина прямоугольника с текстом.

Методы:

affine_transform(trans)	Трансформирует объект исходя из заданных параметров трансформации.
almost_equals(other, tolerance)	Производит примерное сравнения с другой геометрией в пределах заданной точности.
boundary()	Возвращает границы геометрии в виде полилинии.
buffer(distance[, resolution, capStyle, ...])	Производит построение буфера вокруг объекта.
centroid()	Возвращает центроид геометрии.
clone()	Создает копию объекта.
contains(other)	Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.
convex_hull()	Возвращает минимальный окаямляющий полигон со всеми выпуклыми углами.
covers(other)	Возвращает True, если геометрия охватывает геометрию other.
crosses(other)	Возвращает True, если при пересечении геометрий объекты частично пересекаются.
difference(other)	Возвращает область первой геометрии, которая не пересечена второй геометрией.
disjoint(other)	Возвращает True, если геометрии не пересекаются и не соприкасаются.

continues on next page

Таблица 25 - продолжение с предыдущей страницы

<code>envelope()</code>	Возвращает полигон, описывающий заданную геометрию.
<code>equals(other)</code>	Производит сравнение с другой геометрией.
<code>get_area([area_unit, calc_mode])</code>	Рассчитывает площадь, если объект площадной.
<code>get_distance(other[, u])</code>	Производит расчет расстояния до объекта other.
<code>get_length([u, calc_mode])</code>	Рассчитывает длину геометрии.
<code>get_perimeter([u, calc_mode])</code>	Рассчитывает периметр геометрии.
<code>intersection(other)</code>	Возвращает область пересечения с другой геометрией.
<code>intersects(other)</code>	Возвращает True, если геометрии пересекаются.
<code>overlaps(other)</code>	Возвращает True, если пересечение геометрий отличается от обеих геометрий.
<code>relate(other)</code>	Проверяет отношения между объектами.
<code>reproject(cs)</code>	Перепроецирует геометрию в другую систему координат.
<code>rotate(point, angle)</code>	Поворот геометрии относительно заданной точки.
<code>scale(kx, ky)</code>	Масштабирует объект по заданным коэффициентам масштабирования.
<code>shift(dx, dy)</code>	Смещает объект на заданную величину.
<code>size_for_view(style, view)</code>	Размер шрифта в пунктах для конкретного окна.
<code>split_by_polyline(splitter)</code>	Разрезает объект линейным объектом.
<code>symmetric_difference(other)</code>	Возвращает логический XOR областей геометрий (объединение разниц).
<code>to_geojson()</code>	Преобразует геометрию в формат 'GeoJSON'.
<code>to_linestring()</code>	Пробует геометрию преобразовать в линейный объект.
<code>to_mif()</code>	Преобразует геометрию в формат MIF.
<code>to_polygon()</code>	Пробует геометрию преобразовать в площадной объект.
<code>to_wkb()</code>	Возвращает WKB строку для геометрии.
<code>to_wkt()</code>	Возвращает WKT строку для геометрии.
<code>touches(other)</code>	Возвращает True, если геометрии соприкасаются.
<code>union(other)</code>	Возвращает результат объединения двух геометрий.
<code>within(other)</code>	Возвращает True, если геометрия находится полностью внутри геометрии other.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Представление геометрии в виде строки.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(text: str, rect: Union[Rect, QRectF], view: Optional[Union[MapView, ReportView]] = None, angle: float = 0, cs: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **text** – Строка с текстом. Многострочный текст можно задать, вставив символ «\n» в место переноса.
- **rect** – Прямоугольник, в который будет вписан текст. Прямоугольник задается в координатах отчета. Верхней левой точкой является `startPoint`. В этот прямоугольник будет произведена попытка размещения текста.
- **angle** – Угол поворота текстового объекта. Задается значением в градусах против ЧС по отношению к горизонтали.
- **view** – Окно карты или отчета.
- **cs** – Система Координат, в которой создается геометрия.

`__repr__() → str`

Представление геометрии в виде строки.

`affine_transform(trans: QTransform) → Geometry`

Трансформирует объект исходя из заданных параметров трансформации.

Параметры

trans – Матрица трансформации.

См.также:

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

Для выполнения преобразования необходимо сформировать матрицу `PySide2.QtGui.QTransform`. Сделать это можно или последовательно дополняя преобразование или же сразу задав необходимые коэффициенты.

Список 158: Пример последовательного задания матрицы.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
# Формируем матрицу
# Сначала масштабируем
tr = QTransform.fromScale(scale_x, scale_y)
# Затем перемещаем
tr *= QTransform.fromTranslate(dx, dy)
# Применяем матрицу
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

Список 159: Пример задания матрицы через коэффициенты.

```
from PySide2.QtGui import QTransform
dx = 5
dy = 5
scale_x = 0.5
scale_y = 0.5
# Исходная линия
line = Line((1,1), (10,10))
print('source>>', line)
'''
source>> Line begin=(1.0 1.0) end=(10.0 10.0)
'''
# Зададим матрицу, сразу указав коэффициенты
tr = QTransform(scale_x, 0, 0, scale_y, dx, dy)
line_transform = line.affine_transform(tr)
print('transformed>>', line_transform)
'''
transformed>> Line begin=(5.5 5.5) end=(10.0 10.0)
'''
```

almost_equals(other: [Geometry](#), tolerance: [float](#)) → [bool](#)

Производит примерное сравнения с другой геометрией в пределах заданной точности.

Параметры

- **other** – Сравниваемый объект.
- **tolerance** – Точность сравнения.

Результат

Возвращает True, если геометрии равны в пределах заданного отклонения.

property angle: [float](#)

Угол поворота текста, отсчитываемый от горизонтали против часовой стрелки.

boundary() → [Optional\[Geometry\]](#)

Возвращает границы геометрии в виде полилинии.

property bounds: [Rect](#)

Возвращает минимальный ограничивающий прямоугольник.

buffer(distance: `float`, resolution: `int` = 16, capStyle: `LineCapStyle` = `LineCapStyle.Round`, join_style: `LineJoinStyle` = `LineJoinStyle.Round`, mitreLimit: `float` = 5.0) → `Geometry`

Производит построение буфера вокруг объекта.

Параметры

- **distance** - Ширина буфера.
- **resolution** - Количество сегментов на квадрант.
- **capStyle** - Стил ь окончания.
- **join_style** - Стил ь соединения.
- **mitreLimit** - Предел среза.

centroid() → `Point`

Возвращает центроид геометрии.

clone() → `Geometry`

Создает копию объекта.

contains(other: `Geometry`) → `bool`

Возвращает True, если геометрия полностью содержит передаваемую в качестве параметра геометрию.

convex_hull() → `Geometry`

Возвращает минимальный окаймляющий полигон со всеми выпуклыми углами.

property coordsystem: `Optional[CoordSystem]`

Система Координат (СК) геометрии.

covers(other: `Geometry`) → `bool`

Возвращает True, если геометрия охватывает геометрию other.

classmethod create_by_style(text: `str`, point: `Union[Pnt, QPointF]`, style: `Style`, view: `Union[MapView, ReportView]`, angle: `float` = 0, cs: `Optional[CoordSystem]` = None) → `str`

Создает текстовый объект по точке привязки и стиля для карты или отчета. Результирующий объект будет корректен только для текущего состояния окна карты или отчета, куда он должен быть добавлен. Это связано с тем, что расчет размера производится на основании текущего установленного масштаба. Поэтому при изменении масштаба перед добавлением необходимо заново создать текстовый объект.

Параметры

- **text** - Текст.
- **point** - Точка привязки `startPoint`. Этой точкой служит левая верхняя точка.
- **style** - Стил ь оформления текста.
- **view** - Окно карты или отчета.
- **angle** - Угол поворота текстового объекта. Задается значением в градусах против ЧС по отношению к горизонтали.
- **cs** - Система Координат, в которой создается геометрия.

Список 160: Пример изменения параметров текстовых объектов для окна карты.

```
import axipy
from PySide2.QtCore import Qt

mapview = axipy.view_manager.active
if len(axipy.data_manager.tables) and mapview:
    table = axipy.data_manager.tables[0]
    # Новый стиль для текстовых объектов
    new_style = axipy.TextStyle("Droid Sans", 20)
    new_style.color = Qt.red
    new_style.bold = True
    for f in table.items():
        # Для всех текстовых объектов применим данный стиль
        if isinstance(f.geometry, axipy.Text):
            g = f.geometry
            text = axipy.Text.create_by_style(g.text, g.startPoint, style=new_
↪style, view=mapview, angle=g.angle)
            text.endPoint = g.endPoint
            new_style.callout = f.style.callout
            new_style.callout_style = f.style.callout_style
            f.style = new_style
            f.geometry = text
            table.update([f])
    # Сохраним изменения
    table.commit()
```

Список 161: Пример создания текста для отчета.

```
report_view = view_manager.create_reportview()
style_txt = Style.from_mapinfo('Font ("Times New Roman", 0, 23, 16711680)')
text = Text.create_by_style("Пример\ntекста2", (10, 10), style=style_txt,
↪view=report_view, angle=20)
# Поменяем угол
text.angle = -20
# Изменим начальную точку
text.startPoint = (11, 11)
```

crosses(other: [Geometry](#)) → bool

Возвращает True, если при пересечении геометрий объекты частично пересекаются.

difference(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область первой геометрии, которая не пересечена второй геометрией.

disjoint(other: [Geometry](#)) → bool

Возвращает True, если геометрии не пересекаются и не соприкасаются.

static distance_by_points(start: [Union\[Pnt, Tuple\[float, float\]\]](#), end: [Union\[Pnt, Tuple\[float, float\]\]](#), cs: [Optional\[CoordSystem\]](#) = None) → [Tuple](#)

Производит расчет расстояния между двумя точками и азимут от первой до второй точки.

См.также:

Обратная функция `point_by_azimuth()`

Параметры

- **start** – Начальная точка. Если задана СК, то координаты в ней.
- **end** – Конечная точка. Если задана СК, то координаты в ней.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращается пара значений (расстояние в метрах, азимут в градусах)

property endPoint: Pnt

Координаты точки выноски (указки).

envelope() → `Geometry`

Возвращает полигон, описывающий заданную геометрию.

equals(other: `Geometry`) → `bool`

Производит сравнение с другой геометрией.

Параметры

other – Сравниваемая геометрия.

Результат

Возвращает True, если геометрии равны.

static from_geojson(json: `str`, cs: `Optional[CoordSystem]` = None) → `Geometry`

Возвращает геометрию из ее „GeoJSON“ представления.

Параметры

- **json** – json представление в виде строки.
- **cs** – Система координат.

static from_mif(mif: `str`) → `Geometry`

Возвращает геометрию из ее „MIF“ представления.

Параметры

mif – mif представление в виде строки.

classmethod from_wkb(wkb: `bytes`, coordsystem: `Optional[CoordSystem]` = None) → `Geometry`

Создает геометрический объект из строки формата **WKB**.

Параметры

- **wkb** – Строка WKB
- **coordsystem** – Система координат, которая будет установлена для геометрии.

Список 162: Пример.

```
wkb = b'\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00$@\x00\x00\x00\x00\x00\x00'
↳ $@'
pnt = Geometry.from_wkb(wkb)
```

classmethod from_wkt(wkt: str, coordsystem: Optional[CoordSystem] = None) → Geometry

Создает геометрический объект из строки формата WKT.

Параметры

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

Список 163: Пример.

```
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)
```

get_area(area_unit: Optional[AreaUnit] = None, calc_mode: Optional[CalcMode] = None) → float

Рассчитывает площадь, если объект площадной. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере или эллипсоиде по заданному значению calc_mode в квадратных метрах.

Список 164: 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 sq. km:', poly.get_area())
print('Area Mercator sq. m:', poly.get_area(Unit.sq_m))
print('Area Mercator sq. km on plane:', poly.get_area(Unit.sq_km, CalcMode.
↳ PLANE))
print('Perimeter Mercator sq. km:', poly.get_perimeter())
print('Perimeter Mercator sq. 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))
print('Perimeter LL km on ellipsoid:', poly.get_perimeter(Unit.km, CalcMode.
↳ ELLIPSOID))

...
>>> Area Mercator sq. km: 4.017946986687529
>>> Area Mercator sq. m: 4017946.986687529
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> Area Mercator sq. km on plane: 4.0
>>> Perimeter Mercator sq. km: 8.017972048477583
>>> Perimeter Mercator sq. m: 8017.972048477583
>>> Area LL m: 49452172514.0342
>>> Area LL km: 49452.1725140342
>>> Perimeter LL m: 889423.5067063896
>>> Perimeter LL km: 889.4235067063896
>>> Perimeter LL km on ellipsoid: 887.4421376105504
'''
```

Параметры

- **area_unit** – 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.
- **calc_mode** – Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning –

get_distance(other: `Geometry`, u: `Optional[LinearUnit]` = None) → `float`

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

Параметры

- **other** – Анализируемый объект.
- **u** – Единицы измерения, в которых требуется получить результат.

Список 165: 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
'''
```

get_length(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → `float`

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

Список 166: 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))
print('Lenght Mercator km on plane:', ls.get_length(Unit.km, CalcMode.PLANE))
# 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))
print('Lenght LL km on ellipsoid:', ls.get_length(Unit.km, CalcMode.
↳ ELLIPSOID))
...

>>> Length Mercator km: 9.988805508567783
>>> Lenght Mercator m: 9988.805508567782
>>> Lenght Mercator km on plane: 10.0
>>> Length LL m: 1111948.7428468117
>>> Length LL km: 1111.9487428468117
>>> Lenght LL km on ellipsoid: 1113.1949079327358
...

```

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

get_perimeter(u: `Optional[LinearUnit]` = None, calc_mode: `Optional[CalcMode]` = None) → float

Рассчитывает периметр геометрии. Метод применим только для площадных объектов. В случае, если СК задана как Широта/Долгота, то расчет производится на сфере в метрах.

Пример см. `get_area()`

Параметры

- **u** - 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.
- **calc_mode** - Режим расчета. Если не задан, то используется значение по умолчанию для СК геометрии `CoordSystem.supported_calc_modes`.

Исключение

ValueError, RuntimeError, RuntimeWarning -

property height: float

Высота прямоугольника с текстом.

intersection(other: [Geometry](#)) → [Optional\[Geometry\]](#)

Возвращает область пересечения с другой геометрией.

intersects(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии пересекаются.

property is_valid: bool

Проверяет геометрию на валидность.

property is_valid_reason: str

Если геометрия неправильная, возвращает краткую аннотацию причины.

property name: str

Возвращает наименование геометрического объекта.

overlaps(other: [Geometry](#)) → [bool](#)

Возвращает True, если пересечение геометрий отличается от обеих геометрий.

static point_by_azimuth(point: [Union\[Pnt, Tuple\[float, float\]\]](#), azimuth: [float](#), distance: [float](#), cs: [Optional\[CoordSystem\]](#) = None) → [Pnt](#)

Производит расчет координат точки относительно заданной, и находящейся на расстоянии distance по направлению azimuth.

См.также:

Обратная функция [distance_by_points\(\)](#)

Параметры

- **point** – Точка, относительно которой производится расчет. Если задана СК, то в ней.
- **azimuth** – Азимут в градусах, указывающий направление.
- **distance** – Расстояние по азимуту. Задается в метрах.
- **cs** – СК, на базе эллипсоида которой производится расчет. Если не задана, то расчет производится на плоскости.

Результат

Возвращает результирующую точку.

property rect_as_polygon: Polygon

Представление ограничивающего прямоугольника в виде полигона.

relate(other: [Geometry](#)) → [str](#)

Проверяет отношения между объектами.

Подробнее см. [DE-9IM](#)

reproject(cs: [CoordSystem](#)) → [Geometry](#)

Перепроецирует геометрию в другую систему координат.

Параметры

cs – СК, в которой требуется получить объект.

rotate(point: Union[Pnt, Tuple[float, float]], angle: float) → Geometry

Поворот геометрии относительно заданной точки. Поворот производится против часовой стрелки.

Параметры

- **point** – Точка, вокруг которой необходимо произвести поворот.
- **angle** – Угол поворота в градусах.

scale(kx: float, ky: float) → Geometry

Масштабирует объект по заданным коэффициентам масштабирования. После выполнения операции центр результирующего объекта остается прежним.

Параметры

- **kx** – Масштабирование по координате X.
- **ky** – Масштабирование по координате Y.

shift(dx: float, dy: float) → Geometry

Смещает объект на заданную величину. Значения задаются в текущей проекции.

Параметры

- **dx** – Сдвиг по координате X.
- **dy** – Сдвиг по координате Y.

size_for_view(style: TextStyle, view: Union[MapView, ReportView]) → float

Размер шрифта в пунктах для конкретного окна. Значение меняется в зависимости от текущего масштаба.

Параметры

- **style** – Стиль оформления текста.
- **view** – Окно карты или отчета.

split_by_polyline(splitter: Geometry) → Optional[Geometry]

Разрезает объект линейным объектом.

Параметры

splitter – Геометрический объект, которым будет производиться разрезание объекта.

Результат

Возвращает полученный результат

```
poly = Polygon((0,10), (10,10), (10, 0))
line = Line((-5,5), (20,5))
r = poly.split_by_polyline(line)
print(r.wkt)
...
>>> GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5, 5 5, 0 10)), POLYGON
↪((10 5, 10 0, 5 5, 10 5))) GEOMETRYCOLLECTION (POLYGON ((0 10, 10 10, 10 5,
↪5 5, 0 10)), POLYGON ((10 5, 10 0, 5 5, 10 5)))
...
```

property startPoint: Pnt

Координаты точки привязки.

symmetric_difference(other: [Geometry](#)) → [Geometry](#)

Возвращает логический XOR областей геометрий (объединение разниц).

Параметры

other – Геометрия для анализа.

property text: [str](#)

Текст.

to_geojson() → [str](#)

Преобразует геометрию в формат „GeoJSON“.

to_linestring() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в линейный объект.

В случае неудачи возвращает None.

to_mif() → [str](#)

Преобразует геометрию в формат MIF.

```
point = Point(12, 24)
mif = point.to_mif()
```

to_polygon() → [Optional\[Geometry\]](#)

Пробует геометрию преобразовать в площадной объект.

В случае неудачи возвращает None.

to_wkb() → [bytes](#)

Возвращает WKB строку для геометрии.

to_wkt() → [str](#)

Возвращает WKT строку для геометрии.

touches(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрии соприкасаются.

property type: [GeometryType](#)

Возвращает тип геометрического элемента.

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

union(other: [Geometry](#)) → [Geometry](#)

Возвращает результат объединения двух геометрий.

property width: [float](#)

Ширина прямоугольника с текстом.

within(other: [Geometry](#)) → [bool](#)

Возвращает True, если геометрия находится полностью внутри геометрии other.

22.12.17 LineDirection - Направление линии

class ахіру.LineDirection

Базовые классы: `int`, `Enum`

Направление линии.

Атрибуты:

CounterClockwise	Против часовой стрелки.
Clockwise	По часовой стрелке.

22.12.18 LineCapStyle - Стиль окончания линии

class ахіру.LineCapStyle

Стиль окончания линии.

Атрибуты:

Round	Конец линии закруглен.
Flat	Конец линии срезан по ее окончанию.
Square	Конец линии срезан посередине между ее окончанием и полученным результатом.

22.12.19 LineJoinStyle - Стиль соединения линий

class ахіру.LineJoinStyle

Стиль соединения линий.

Атрибуты:

Round	Соединение линий скруглено.
Mitre	Соединение линий плоское.
Bevel	Соединение - точка пересечения края результата

22.12.20 GeometryType - Тип геометрического элемента

class ахіру.GeometryType

Тип геометрического элемента.

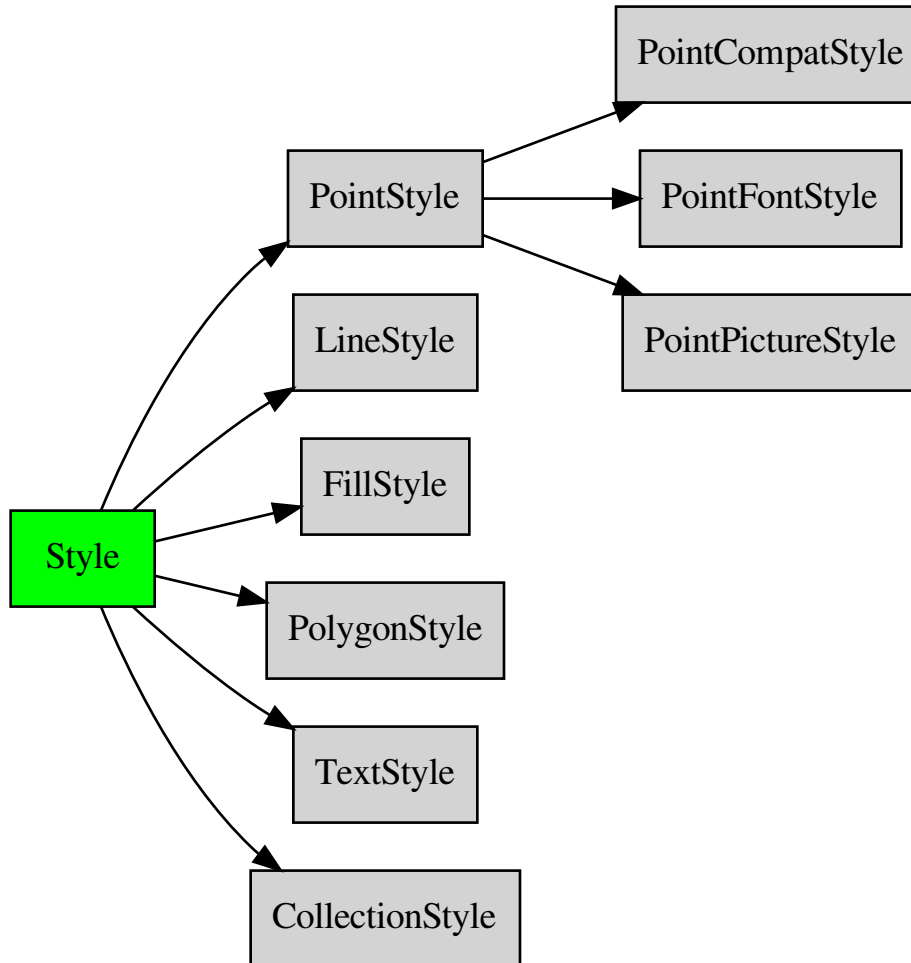
Атрибуты:

Unknown	Не определен
Point	Точка
Line	Линия
LineString	Полилиния
Polygon	Полигон
MultiPoint	Коллекция точек
MultiLineString	Коллекция полилиний
MultiPolygon	Коллекция полигонов
GeometryCollection	Смешанная коллекция
Arc	Дуга
Ellipse	Эллипс
Rectangle	Прямоугольник
RoundedRectangle	Скругленный прямоугольник
Text	Текст

22.13 Style - Стил

22.13.1 Style - Стил

Иерархия классов стилей геометрических объектов:



class ахіру.Style

Абстрактный класс стиля оформления геометрического объекта. Определяет как будет отображен геометрический объект.

Примечание: Для получения текстового представления стиля можно воспользоваться функцией `str`.

Классовые методы:

<code>for_geometry()</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type()</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone() → Style`

Создаёт копию объекта стиля.

`draw(geometry: Geometry, painter: QPainter)`

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 167: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

```
classmethod for_geometry(geom: Union[Point, MultiPoint]) → PointStyle
classmethod for_geometry(geom: Union[Line, LineString, Arc, MultiLineString]) →
    LineStyle
classmethod for_geometry(geom: Union[Polygon, Rectangle, RoundRectangle,
    Ellipse, MultiPolygon]) → PolygonStyle
classmethod for_geometry(geom: GeometryCollection) → CollectionStyle
classmethod for_geometry(geom: Text) → TextStyle
classmethod for_geometry(geom: Geometry) → Style
```

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

```
classmethod for_geometry_type(geometry_type:
    ~typing.Literal[<StyleGeometryType.Point: 1>]) →
    PointStyle
classmethod for_geometry_type(geometry_type:
    ~typing.Literal[<StyleGeometryType.Linear: 2>])
    → LineStyle
classmethod for_geometry_type(geometry_type:
    ~typing.Literal[<StyleGeometryType.Polygonal:
    3>]) → PolygonStyle
classmethod for_geometry_type(geometry_type:
    ~typing.Literal[<StyleGeometryType.MultiGeometry:
    4>]) → CollectionStyle
classmethod for_geometry_type(geometry_type:
    ~typing.Literal[<StyleGeometryType.Text: 5>]) →
    TextStyle
classmethod for_geometry_type(geometry_type: StyleGeometryType) → Style
```

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

```
classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]
```

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
...
>>> Pen (1, 2, 0) Brush (8, 255)
...
```

22.13.2 StyleGeometryType - Вид геометрического объекта

class axipy.StyleGeometryType

Тип геометрического объекта для стиля.

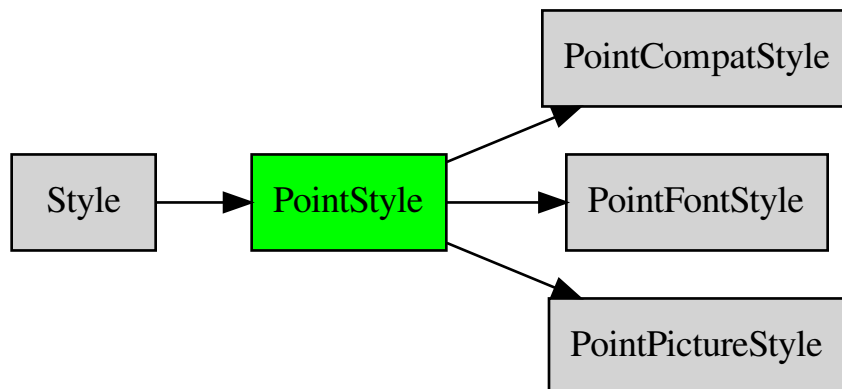
Используется как категория геометрии при выборе стиля представления этой геометрии.

Атрибуты:

Point	Точечные.
Linear	Линейные.
Polygonal	Полигональные.
MultiGeometry	Разнородная коллекция.
Text	Текстовые.

22.13.3 PointStyle - Стиль точек

Иерархия классов стилей геометрических объектов:



class ахіру.PointStyle

Базовые классы: [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\(\)](#).

Список 168: Пример.

```
style_1 = PointStyle.create_mi_compat(color=Qt.blue)
style_2 = PointStyle.create_mi_font(42, Qt.red, 24)
style_3 = PointStyle.create_mi_picture('AMBU-64.bmp')
```

Классовые методы:

create_mi_compat ([symbol, color, pointSize])	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle .
create_mi_font ([symbol, color, size, ...])	Создание стиля на базе шрифта True Type PointFontStyle .
create_mi_picture (filename[, color, size, ...])	Создание стиля со ссылкой на растровый файл PointPictureStyle .
for_geometry (geom)	Возвращает стиль по умолчанию из настроек для переданного объекта.
for_geometry_type (geometry_type)	Возвращает стиль по умолчанию для переданного типа геометрии.
from_mapinfo (mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

color	Устанавливает или возвращает цвет символа.

Методы:

clone ()	Создаёт копию объекта стиля.
draw (geometry, painter)	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
to_mapinfo ()	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone() → Style`

Создаёт копию объекта стиля.

property color: QColor

Устанавливает или возвращает цвет символа.

static create_mi_compat(symbol: int = 35, color: Union[QColor, GlobalColor] = Qt.red, pointSize: int = 8) → PointCompatStyle

Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle.

Параметры

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

static create_mi_font(symbol: int = 36, color: Union[QColor, GlobalColor] = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: int = 0, rotation: float = 0.0) → PointFontStyle

Создание стиля на базе шрифта True Type PointFontStyle.

Параметры

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

```
static create_mi_picture(filename: str, color: Union[QColor, GlobalColor] =
                        Qt.black, size: int = 12, customstyle: int = 0) →
                        PointPictureStyle
```

Создание стиля со ссылкой на растровый файл `PointPictureStyle`.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находиться в каталоге CustSymb с ресурсами. Например, "Arrow.BMP".
- **color** – Цвет символа.
- **size** – Размер символа в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

```
draw(geometry: Geometry, painter: QPainter)
```

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 169: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

```
classmethod for_geometry(geom: Geometry) → Style
```

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

```
classmethod for_geometry_type(geometry_type: StyleGeometryType) → Style
```

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

```
classmethod from_mapinfo(mapbasic_string: str) → Optional[Style]
```

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
...
>>> Pen (1, 2, 0) Brush (8, 255)
...
```

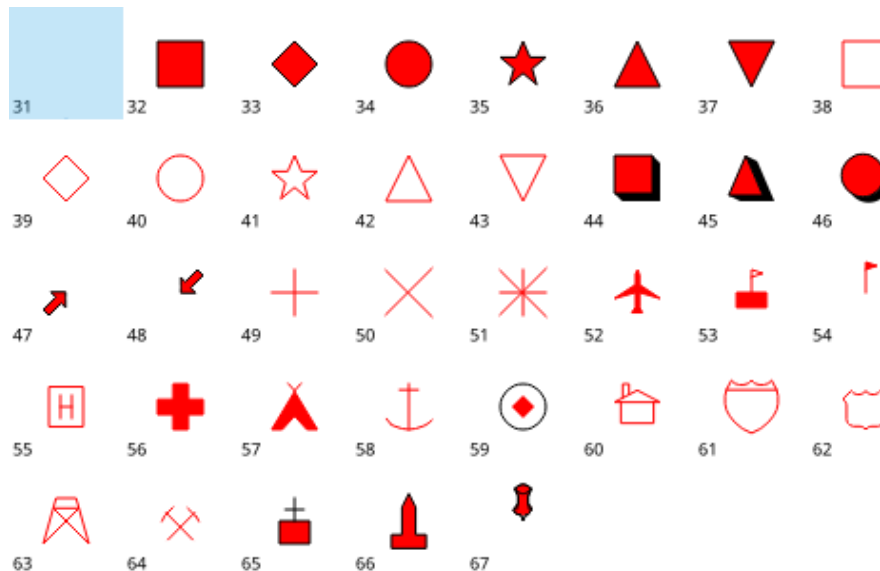
22.13.4 PointCompatStyle - Стил, сумісний з MapInfo 3

class ахіру.PointCompatStyle

Базовые классы: [PointStyle](#)

Класс стили, сумісного з MapInfo 3.

В системе доступны следующие стили:



Список 170: Пример.

```
style = PointCompatStyle()
style.color = Qt.blue
style.symbol = 43
```

Конструктор класса:

`__init__([symbol, color, pointSize])` Конструктор класса.

Классовые методы:

<code>create_mi_compat([symbol, color, pointSize])</code>	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle .
<code>create_mi_font([symbol, color, size, ...])</code>	Создание стиля на базе шрифта True Type PointFontStyle .
<code>create_mi_picture(filename[, color, size, ...])</code>	Создание стиля со ссылкой на растровый файл PointPictureStyle .
<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>color</code>	Устанавливает или возвращает цвет символа.
<code>size</code>	Устанавливает или возвращает размер символа в пунктах.
<code>symbol</code>	Устанавливает или возвращает номер символа.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(symbol: int = 35, color: Union[QColor, GlobalColor] = Qt.red, pointSize: int = 8)`

Конструктор класса.

Параметры

- **symbol** – Номер символа, который будет отображен. Для создания невидимого символа используйте значение 31. Стандартный набор условных знаков включает символы от 31 до 67.

- **color** - Цвет символа.
- **pointSize** - Целое число, размер символа в пунктах от 1 до 48.

`__repr__()` → `str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone()` → `Style`

Создаёт копию объекта стиля.

property color: QColor

Устанавливает или возвращает цвет символа.

static create_mi_compat(symbol: `int` = 35, color: `Union[QColor, GlobalColor]` = `Qt.red`, pointSize: `int` = 8) → `PointCompatStyle`

Создание стиля в виде совместимого с MapInfo 3 `PointCompatStyle`.

Параметры

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

static create_mi_font(symbol: `int` = 36, color: `Union[QColor, GlobalColor]` = `Qt.red`, size: `int` = 8, fontname: `str` = 'Axioma MI MapSymbols', fontStyle: `int` = 0, rotation: `float` = 0.0) → `PointFontStyle`

Создание стиля на базе шрифта True Type `PointFontStyle`.

Параметры

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

static create_mi_picture(filename: `str`, color: `Union[QColor, GlobalColor]` = `Qt.black`, size: `int` = 12, customstyle: `int` = 0) → `PointPictureStyle`

Создание стиля со ссылкой на растровый файл `PointPictureStyle`.

Параметры

- **filename** - Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находиться в каталоге CustSymb с ресурсами. Например, "Arrow.BMP".
- **color** - Цвет символа.
- **size** - Размер символа в пунктах от 1 до 48.
- **customstyle** - Задание дополнительных параметров стиля оформления.

draw(geometry: [Geometry](#), painter: [QPainter](#))

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** - Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** - Контекст вывода.

Список 171: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom - Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type - Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string - Строка в формате MapBasic.

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

property size: [int](#)

Устанавливает или возвращает размер символа в пунктах.

property symbol: int

Устанавливает или возвращает номер символа.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

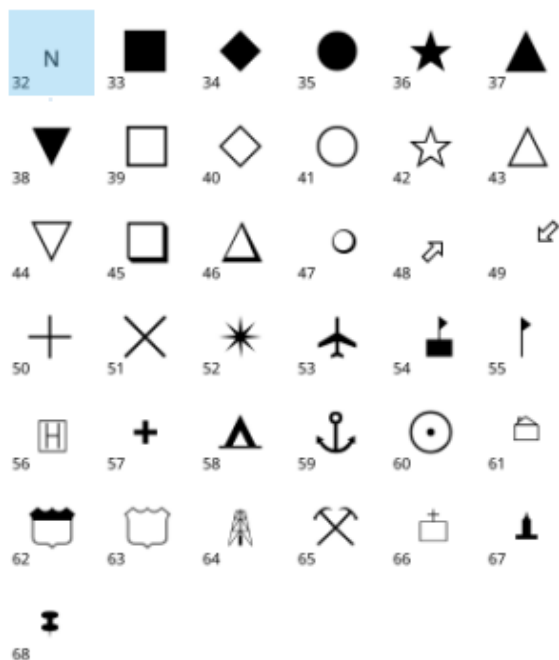
22.13.5 PointFontStyle - Стил на базе шрифта True Type

class ахіру.PointFontStyle

Базовые классы: [PointStyle](#)

Стил на базе шрифта True Type.

В системе доступны следующие стили:



Список 172: Пример.

```
fs = PointFontStyle()
fs.font_name = 'Axioma MI Oil&Gas'
fs.has_shadow = True
fs.rotation = 30
fs.bold = True
fs.size = 44
fs.symbol = 45
fs.black_border = True
fs.white_border = True
```

Конструктор класса:

<code>__init__([symbol, color, size, fontname, ...])</code>	Конструктор класса.

Классовые методы:

<code>create_mi_compat([symbol, color, pointSize])</code>	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle .
<code>create_mi_font([symbol, color, size, ...])</code>	Создание стиля на базе шрифта True Type PointFontStyle .
<code>create_mi_picture(filename[, color, size, ...])</code>	Создание стиля со ссылкой на растровый файл PointPictureStyle .
<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>black_border</code>	Устанавливает или возвращает признак 'Темная окантовка'.
<code>bold</code>	Устанавливает или возвращает признак 'Жирности шрифта'.
<code>color</code>	Устанавливает или возвращает цвет символа.
<code>font_name</code>	Устанавливает или возвращает наименование шрифта.
<code>has_shadow</code>	Устанавливает или возвращает признак 'Тень'.
<code>rotation</code>	Устанавливает или возвращает угол поворота.
<code>size</code>	Устанавливает или возвращает размер символа в пунктах.
<code>symbol</code>	Устанавливает или возвращает номер символа.
<code>white_border</code>	Устанавливает или возвращает признак 'Светлая окантовка'.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

Классы:

<code>FontStyleTypes</code>	Тип шрифта.

class FontStyleTypes

Базовые классы: `int`, `Enum`

Тип шрифта.

Атрибуты:

<code>Plain</code>	Обычный текст.
<code>Bold</code>	Жирный текст.
<code>Outline</code>	Черная кайма вокруг символа.
<code>Shadow</code>	Тень.
<code>Halo</code>	Белая кайма вокруг символа.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(symbol: int = 36, color: Union[QColor, GlobalColor] = Qt.red, size: int = 8, fontname: str = 'Axioma MI MapSymbols', fontstyle: FontStyleTypes = FontStyleTypes.Plain, rotation: float = 0.0)`

Конструктор класса.

Параметры

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

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property black_border: bool

Устанавливает или возвращает признак „Темная окантовка“.

property bold: bool

Устанавливает или возвращает признак „Жирности шрифта“.

clone() → [Style](#)

Создаёт копию объекта стиля.

property color: QColor

Устанавливает или возвращает цвет символа.

static create_mi_compat(symbol: [int](#) = 35, color: [Union](#)[[QColor](#), [GlobalColor](#)] = [Qt.red](#), pointSize: [int](#) = 8) → [PointCompatStyle](#)

Создание стиля в виде совместимого с MapInfo 3 [PointCompatStyle](#).

Параметры

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

static create_mi_font(symbol: [int](#) = 36, color: [Union](#)[[QColor](#), [GlobalColor](#)] = [Qt.red](#), size: [int](#) = 8, fontname: [str](#) = 'Axioma MI MapSymbols', fontstyle: [int](#) = 0, rotation: [float](#) = 0.0) → [PointFontStyle](#)

Создание стиля на базе шрифта True Type [PointFontStyle](#).

Параметры

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

static create_mi_picture(filename: [str](#), color: [Union](#)[[QColor](#), [GlobalColor](#)] = [Qt.black](#), size: [int](#) = 12, customstyle: [int](#) = 0) → [PointPictureStyle](#)

Создание стиля со ссылкой на растровый файл [PointPictureStyle](#).

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находиться в каталоге CustSymb с ресурсами. Например, „Arrow.BMP“.
- **color** – Цвет символа.

- **size** – Размер символа в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

draw(geometry: [Geometry](#), painter: [QPainter](#))

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 173: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property font_name: str

Устанавливает или возвращает наименование шрифта.

Например, Adobe Helvetica.

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

property has_shadow: bool

Устанавливает или возвращает признак „Тень“.

property rotation: float

Устанавливает или возвращает угол поворота.

property size: int

Устанавливает или возвращает размер символа в пунктах.

property symbol: int

Устанавливает или возвращает номер символа.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

property white_border: bool

Устанавливает или возвращает признак „Светлая окантовка“.

22.13.6 PointPictureStyle - Стиль со ссылкой на растровый файл

class axipy.PointPictureStyle

Базовые классы: [PointStyle](#)

Стиль со ссылкой на растровый файл.

Таблица 26: 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“.

Список 174: Пример.

```
fs = PointPictureStyle('AMBU1-32.bmp')
fs.color = Qt.red
fs.apply_color = True
fs.actual_size = True
fs.show_background = True
```

Конструктор класса:

```
__init__(filename[, color, size, Конструктор класса.
customstyle])
```

Классовые методы:

<code>create_mi_compat([symbol, color, pointSize])</code>	Создание стиля в виде совместимого с MapInfo 3 PointCompatStyle .
<code>create_mi_font([symbol, color, size, ...])</code>	Создание стиля на базе шрифта True Type PointFontStyle .
<code>create_mi_picture(filename[, color, size, ...])</code>	Создание стиля со ссылкой на растровый файл PointPictureStyle .
<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>actual_size</code>	Устанавливает или возвращает признак 'Реальный размер'.
<code>apply_color</code>	Устанавливает или возвращает признак 'Применить цвет'.
<code>color</code>	Устанавливает или возвращает цвет символа.
<code>filename</code>	Устанавливает или возвращает наименование файла раstra.
<code>show_background</code>	Устанавливает или возвращает признак 'Непрозрачный фон'.
<code>size</code>	Устанавливает или возвращает размер символа в пунктах.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__init__(filename: str, color: Union[QColor, GlobalColor] = Qt.black, size: int = 12, customstyle: int = 0)`

Конструктор класса.

Параметры

- **filename** – Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находиться в каталоге CustSymb с ресурсами. Например, "Arrow.BMP".
- **color** – Цвет символа.
- **size** – Размер символа в пунктах от 1 до 48.
- **customstyle** – Задание дополнительных параметров стиля оформления.

`__repr__()` → `str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property actual_size: bool

Устанавливает или возвращает признак „Реальный размер“.

property apply_color: bool

Устанавливает или возвращает признак „Применить цвет“.

`clone()` → `Style`

Создаёт копию объекта стиля.

property color: QColor

Устанавливает или возвращает цвет символа.

static create_mi_compat(symbol: `int` = 35, color: `Union[QColor, GlobalColor]` = Qt.red, pointSize: `int` = 8) → `PointCompatStyle`

Создание стиля в виде совместимого с MapInfo 3 `PointCompatStyle`.

Параметры

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

static create_mi_font(symbol: `int` = 36, color: `Union[QColor, GlobalColor]` = Qt.red, size: `int` = 8, fontname: `str` = 'Axioma MI MapSymbols', fontstyle: `int` = 0, rotation: `float` = 0.0) → `PointFontStyle`

Создание стиля на базе шрифта True Type `PointFontStyle`.

Параметры

- **symbol** – Целое, имеющее значение 31 или больше, определяющее, какой используется символ из шрифтов TrueType. Для создания невидимого символа используйте значение 31.
- **color** – Цвет символа
- **size** – Целое число, размер символа в пунктах от 1 до 48;

- **fontname** - Строка с именем шрифта TrueType (например, значение по умолчанию „Axioma MI MapSymbols“)
- **fontstyle** - Стиль дополнительного оформления, например, курсивный текст. Возможные параметры см. в таблице ниже. Для указания нескольких параметров их суммируют между собой.
- **rotation** - Угол поворота символа в градусах.

static create_mi_picture(filename: [str](#), color: [Union\[QColor, GlobalColor\]](#) = [Qt.black](#), size: [int](#) = 12, customstyle: [int](#) = 0) → [PointPictureStyle](#)

Создание стиля со ссылкой на растровый файл [PointPictureStyle](#).

Параметры

- **filename** - Наименование растрового файла. Строка до 31 символа длиной. Данный файл должен находиться в каталоге CustSymb с ресурсами. Например, „Arrow.BMP“.
- **color** - Цвет символа.
- **size** - Размер символа в пунктах от 1 до 48.
- **customstyle** - Задание дополнительных параметров стиля оформления.

draw(geometry: [Geometry](#), painter: [QPainter](#))

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** - Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** - Контекст вывода.

Список 175: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property filename: [str](#)

Устанавливает или возвращает наименование файла раstra.

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

- **geom** - Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod `for_geometry_type(geometry_type: StyleGeometryType) → Style`

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type - Тип геометрического объекта, для которого необходимо получить стиль.

classmethod `from_mapinfo(mapbasic_string: str) → Optional[Style]`

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string - Строка в формате MapBasic.

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

property `show_background: bool`

Устанавливает или возвращает признак „Непрозрачный фон“.

property `size: int`

Устанавливает или возвращает размер символа в пунктах.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

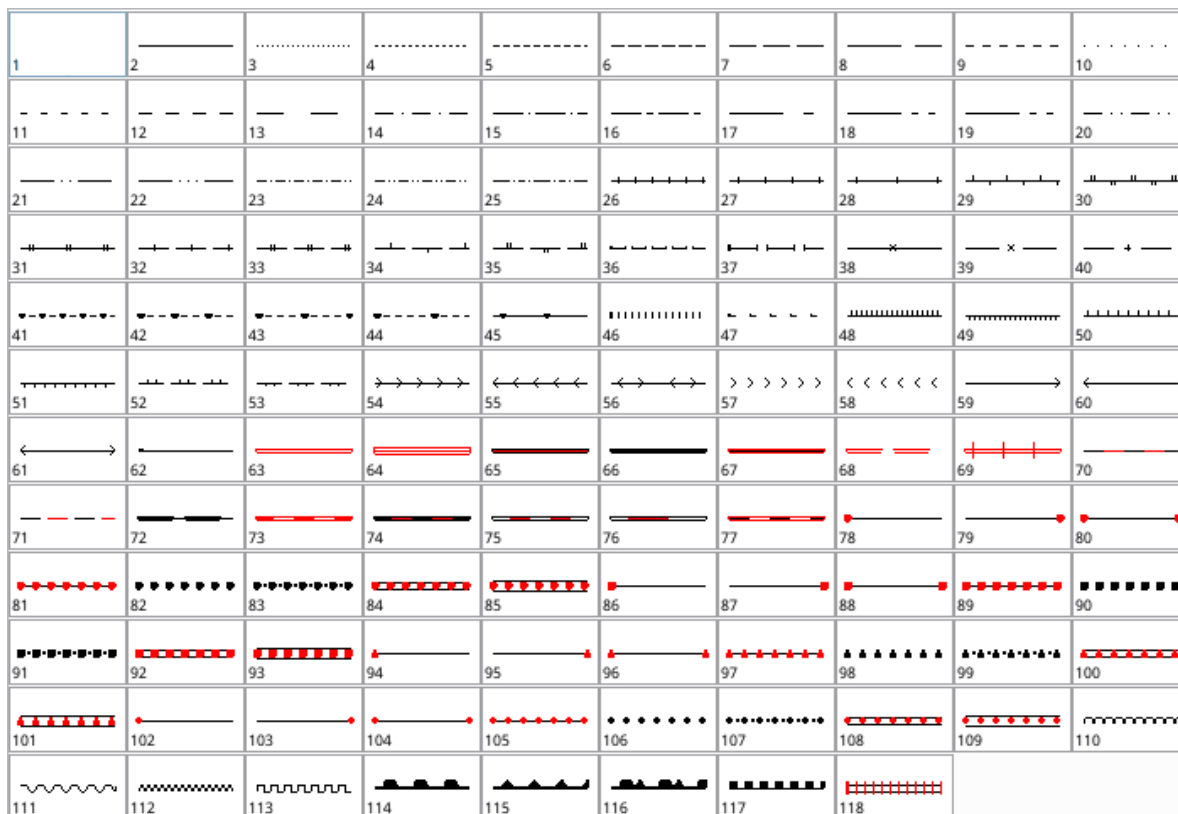
22.13.7 LineStyle - Стиль линий

class `axipy.LineStyle`

Базовые классы: `Style`

Стиль линейного объекта, совместимый с `MapInfo`.

В системе доступны следующие стили линии:



Список 176: Пример.

```
style = LineStyle(3, Qt.red)
```

Конструктор класса:

<code>__init__([pattern, color, width])</code>	Конструктор класса.

Классовые методы:

<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>color</code>	Устанавливает или возвращает цвет линии.
<code>pattern</code>	Устанавливает или возвращает номер стиля линии.
<code>width</code>	Устанавливает или возвращает толщину линии.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__init__(pattern: int = 2, color: Union[QColor, GlobalColor] = Qt.black, width: int = 1)`
Конструктор класса.

Параметры

- **pattern** - Тип линии. Типы линий обозначаются кодами от 1 до 118. Тип 1 представляет собой невидимую линию.
- **color** - Цвет линии
- **width** - 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.

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone() → Style`
Создаёт копию объекта стиля.

property color: QColor
Устанавливает или возвращает цвет линии.

draw(geometry: Geometry, painter: QPainter)
Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 177: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

property pattern: int

Устанавливает или возвращает номер стиля линии.

to_mapinfo() → [str](#)

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

property width: int

Устанавливает или возвращает толщину линии.

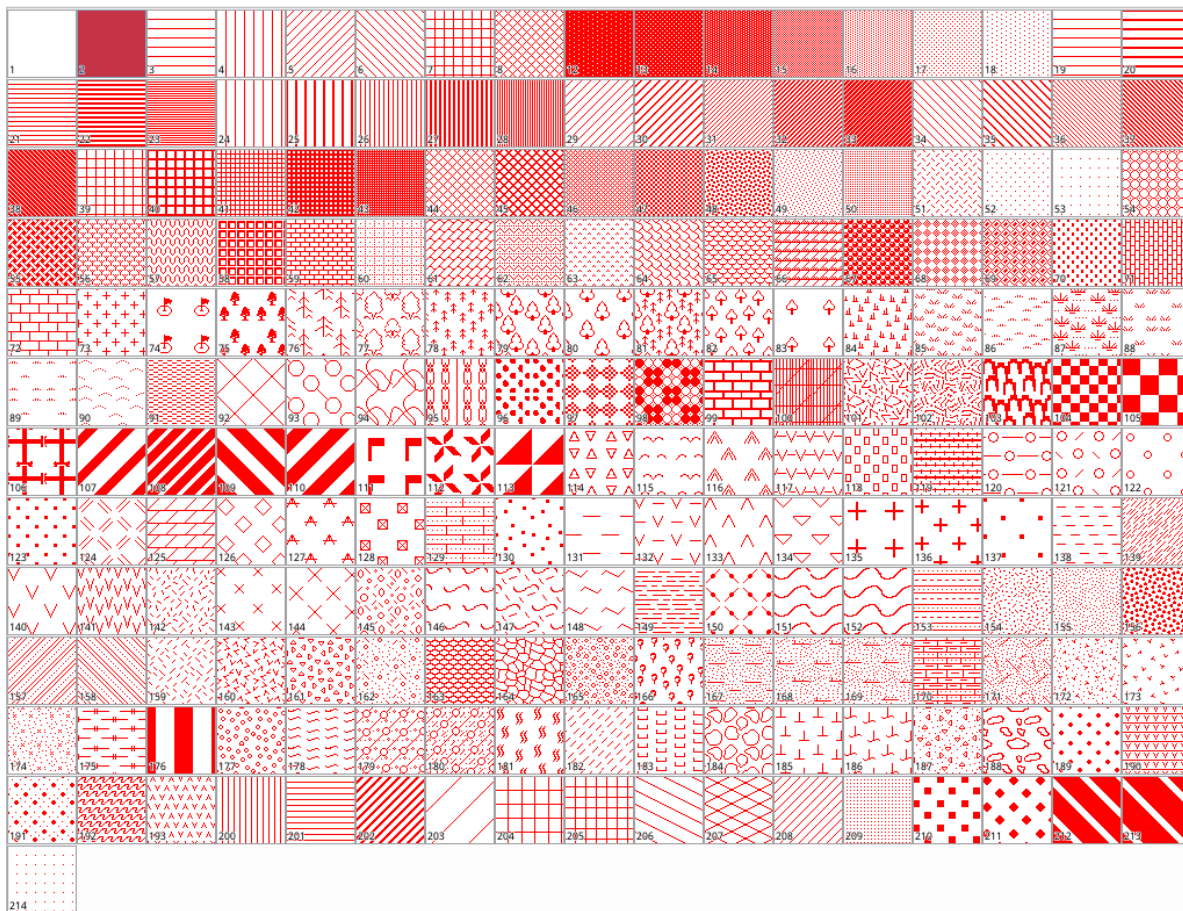
22.13.8 FillStyle - Стиль заливки полигона

class axipy.FillStyle

Базовые классы: [Style](#)

Стиль заливки полигонов [PolygonStyle](#).

В системе доступны следующие стили заливки:



Конструктор класса:

<code>__init__([pattern, color])</code>	Конструктор класса.

Классовые методы:

<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>bg_color</code>	Устанавливает или возвращает цвет фона.
<code>color</code>	Устанавливает или возвращает цвет линии.
<code>pattern</code>	Устанавливает или возвращает номер стиля заливки.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(pattern: int = 1, color: Union[QColor, GlobalColor] = Qt.transparent)`

Конструктор класса.

Параметры

- **pattern** – Номер стиля заливки.
- **color** – Цвет основной заливки.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property bg_color: QColor

Устанавливает или возвращает цвет фона.

`clone() → Style`

Создаёт копию объекта стиля.

property color: QColor

Устанавливает или возвращает цвет линии.

draw(geometry: [Geometry](#), painter: [QPainter](#))

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 178: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

property pattern: [int](#)

Устанавливает или возвращает номер стиля заливки.

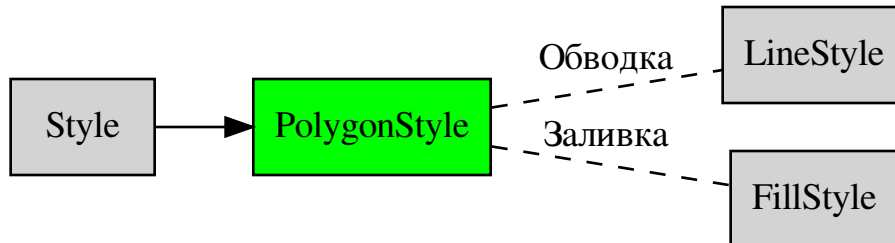
to_mapinfo() → [str](#)

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.9 PolygonStyle - Стиль полигонов

Зависимости стиля площадного объекта:



class ахіру.**PolygonStyle**

Базовые классы: **Style**

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

Список 179: 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())
# Установим параметры заливки через свойства
plstyle.fill.pattern = 5
plstyle.fill.color = Qt.blue
# Установим параметры обводки через свойства
plstyle.border.width = 4
plstyle.border.color = Qt.blue
'''
>>> Brush (1, 16777215)
>>> Brush (1, 65280, 255)
>>> Pen (1, 2, 0) Brush (1, 65280, 255)
'''
  
```

Конструктор класса:

`__init__`([pattern, color, pattern_pen]) Конструктор класса.

Классовые методы:

<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>border</code>	Стиль окантовки.
<code>fill</code>	Стиль заливки.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>set_brush([pattern, color, bgColor])</code>	Задание стиля заливки площадного объекта.
<code>set_pen([pattern, color, width])</code>	Задание стиля обводки.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(pattern: int = 1, color: Union[QColor, GlobalColor] = Qt.white, pattern_pen: int = 1)`

Конструктор класса.

Параметры

- **pattern** - Номер стиля заливки.
- **color** - Цвет основной заливки.
- **pattern_pen** - Цвет обводки. По умолчанию обводка отсутствует.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property border: [Optional\[LineStyle\]](#)

Стиль окантовки.

Может отсутствовать. Для переопределения можно также использовать [set_pen\(\)](#).

clone() → [Style](#)

Создаёт копию объекта стиля.

draw(geometry: [Geometry](#), painter: [QPainter](#))

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 180: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property fill: [FillStyle](#)

Стиль заливки.

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

```
set_brush(pattern: int = 1, color: Union[QColor, GlobalColor] = Qt.white, bgColor:
Union[QColor, GlobalColor] = Qt.transparent)
```

Задание стиля заливки площадного объекта.

Параметры

- **pattern** - 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** - Цвет основной заливки.
- **bgColor** - Цвет заднего фона, если заливка неполная.

Доступные стили заливки см. [FillStyle](#):

```
set_pen(pattern: int = 2, color: Union[QColor, GlobalColor] = Qt.black, width: int = 1)
Задание стиля обводки. Параметры аналогичны при задании стиля линии
LineStyle()
```

Параметры

- **pattern** - Номер стиля линии.
- **color** - Цвет линии
- **width** - Толщина линии.

```
to_mapinfo() → str
```

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.13.10 TextStyle - Стил ь текста

```
class axipy.TextStyle
```

Базовые классы: [Style](#)

Стил ь текстового объекта.

Таблица 27: 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

Список 181: Пример.

```
style = TextStyle("Droid Sans", 24)
style.fontname = 'Adobe Helvetica'
style.color = Qt.red
style.bold = True
style.italic = True
style.shadow = True
style.spacing = True
style.capital = True
style.alignment = TextAlignment.Center
print(style.size)
style.size = 22
style.callout = TextCallout.Arrow
arrow_style = LineStyle(2, Qt.red)
style.callout_style = arrow_style
print(style.to_mapinfo())
'''
>>> Font ("Adobe Helvetica", 1571, 22, 16711680) Label Line Arrow Line (1, 2, ↪
↪16711680)
'''
```

Примечание: При назначении стиля для текста необходимо помнить, что его параметры и параметры геометрии `ахіру.Text` взаимосвязаны.

Конструктор класса:

<code>__init__(fontname, size[, style, Конструктор класса. forecolor, ...])</code>

Классовые методы:

<code>for_geometry(geom)</code>	Возвращает стиль по умолчанию из настроек для переданного объекта.
<code>for_geometry_type(geometry_type)</code>	Возвращает стиль по умолчанию для переданного типа геометрии.
<code>from_mapinfo(mapbasic_string)</code>	Получает стиль из строки формата MapBasic.

Свойства:

<code>alignment</code>	Устанавливает или возвращает выравнивание текста.
<code>bg_color</code>	Устанавливает или возвращает цвет фона текста.
<code>bg_type</code>	Устанавливает или возвращает тип отрисовки фона текста.
<code>bold</code>	Устанавливает или возвращает признак 'Жирный текст'.
<code>callout</code>	Устанавливает или возвращает тип выноски.
<code>callout_style</code>	Устанавливает или возвращает стиль выноски.
<code>capital</code>	Устанавливает или возвращает признак 'Все заглавные'.
<code>color</code>	Устанавливает или возвращает цвет текста.
<code>fontname</code>	Устанавливает или возвращает наименование шрифта.
<code>italic</code>	Устанавливает или возвращает признак 'Курсив текста'.
<code>shadow</code>	Устанавливает или возвращает признак 'Тень'.
<code>size</code>	Устанавливает или возвращает базовый размер шрифта в пунктах.
<code>spacing</code>	Устанавливает или возвращает признак 'Разрядка'.
<code>underline</code>	Устанавливает или возвращает признак 'Подчеркивание'.

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__init__(fontname: str, size: int, style: int = 0, forecolor: Union[QColor, GlobalColor] = Qt.black, bgcolor: Union[QColor, GlobalColor] = Qt.transparent)`

Конструктор класса.

Параметры

- **fontname** – Наименование шрифта.
- **size** – 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** – Дополнительные параметры стиля. Подробнее см. в таблице ниже. Стоит заметить, что если оставить значение, равным 0, то необходимые свойства можно установить позже через соответствующие свойства.
- **forecolor** – Цвет шрифта.
- **backcolor** – Цвет заднего фона, если он задан.

`__repr__()` → `str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__()` → `str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property alignment: `TextAlignment`

Устанавливает или возвращает выравнивание текста.

property bg_color: `QColor`

Устанавливает или возвращает цвет фона текста.

property bg_type: `TextBackgroundType`

Устанавливает или возвращает тип отрисовки фона текста.

property bold: `bool`

Устанавливает или возвращает признак „Жирный текст“.

property callout: `TextCallout`

Устанавливает или возвращает тип выноски.

property callout_style: `LineStyle`

Устанавливает или возвращает стиль выноски.

property capital: `bool`

Устанавливает или возвращает признак „Все заглавные“.

`clone()` → `Style`

Создаёт копию объекта стиля.

property color: `QColor`

Устанавливает или возвращает цвет текста.

draw(geometry: `Geometry`, painter: `QPainter`)

Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** – Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** – Контекст вывода.

Список 182: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

property fontname: str

Устанавливает или возвращает наименование шрифта.

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom – Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type – Тип геометрического объекта, для которого необходимо получить стиль.

classmethod from_mapinfo(mapbasic_string: str) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string – Строка в формате MapBasic.

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

property italic: bool

Устанавливает или возвращает признак „Курсив текста“.

property shadow: bool

Устанавливает или возвращает признак „Тень“.

property size: float

Устанавливает или возвращает базовый размер шрифта в пунктах.

Точный размер шрифта высчитывается исходя из контекста рисования (карта или отчет).

property spacing: bool

Устанавливает или возвращает признак „Разрядка“.

to_mapinfo() → str

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
...
>>> Pen (1, 2, 0) Brush (8, 255)
...
```

property underline: bool

Устанавливает или возвращает признак „Подчеркивание“.

22.13.11 TextCallout - Тип выноски

class axipy.TextCallout

Тип выноски.

Атрибуты:

Arrow	Стрелка.
Line	Линия.
NoCallout	Не отображать.

22.13.12 TextAlignment - Выравнивание текста

class axipy.TextAlignment

Выравнивание текста.

Атрибуты:

Center	По центру.
Left	По левому краю.
Right	По правому краю.

22.13.13 TextBackgroundType - Тип отрисовки фона

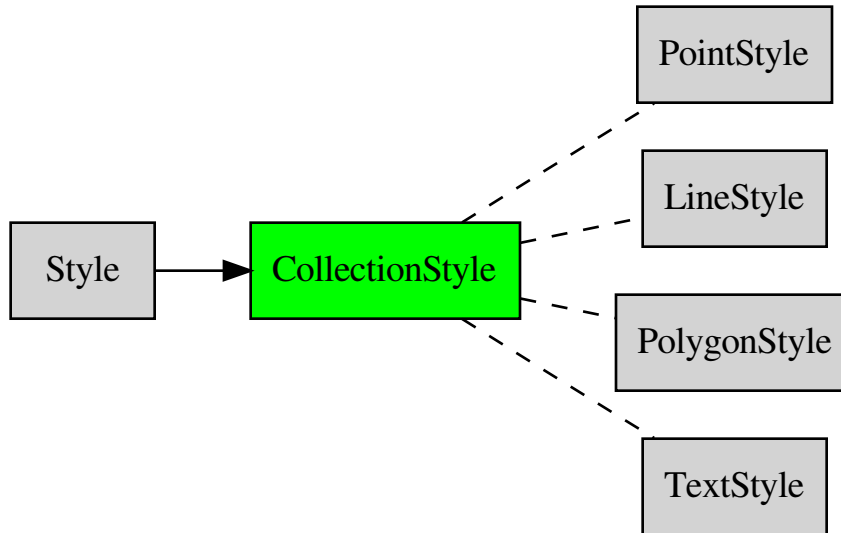
class axipy.TextBackgroundType

Тип отрисовки фона текстового объекта.

Атрибуты:

Frame	Рамка вокруг текста.
NoBackground	Фон отсутствует.
Outline	Обводка по контуру букв текста.

22.13.14 CollectionStyle - Стиль коллекций



class axipy.CollectionStyle

Базовые классы: [Style](#)

Смешанный стиль для разнородного типа объектов.

Контейнер стилей. может применяться вкупе с геометрическим объектом типа разнородная коллекция [axipy.GeometryCollection](#). Для задания или переопределения стилей простейших объектов, необходимо вызывать соответствующие методы для необходимых типов объектов.

Примечание: Объекты стилей, полученные через методы [line\(\)](#), [polygon\(\)](#) и т.д. будут удалены сразу после удаления объекта стиля коллекции. Если их нужно сохранить, воспользуйтесь операцией [axipy.Style.clone\(\)](#).

Классовые методы:

for_geometry(geom)	Возвращает стиль по умолчанию из настроек для переданного объекта.
for_geometry_type(geometry_type)	Возвращает стиль по умолчанию для переданного типа геометрии.
from_mapinfo(mapbasic_string)	Получает стиль из строки формата MapBasic.

Свойства:

<code>line</code>	Стиль для линейных объектов LineStyle .
<code>point</code>	Стиль для точечных объектов PointStyle .
<code>polygon</code>	Стиль для полигональных объектов PolygonStyle .
<code>text</code>	Стиль для текстовых объектов TextStyle .

Методы:

<code>clone()</code>	Создаёт копию объекта стиля.
<code>draw(geometry, painter)</code>	Рисует геометрический объект с текущим стилем в произвольном контексте вывода.
<code>find_style(geom)</code>	Возвращает стиль, подходящий для переданной геометрии.
<code>for_line(style)</code>	Задание стиля для линейных объектов LineStyle .
<code>for_point(style)</code>	Задание стиля для точечных объектов PointStyle .
<code>for_polygon(style)</code>	Задание стиля для полигональных объектов PolygonStyle .
<code>for_text(style)</code>	Задание стиля для текстовых объектов TextStyle .
<code>to_mapinfo()</code>	Возвращает строковое представление в формате MapBasic.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`clone() → Style`
Создаёт копию объекта стиля.

`draw(geometry: Geometry, painter: QPainter)`
Рисует геометрический объект с текущим стилем в произвольном контексте вывода. Это может быть востребовано при желании отрисовать геометрию со стилем на форме или диалоге.

Параметры

- **geometry** - Геометрия. Должна соответствовать стилю. Т.е. если объект полигон, а стиль для рисования точечных объектов, то ничего нарисовано не будет.
- **painter** - Контекст вывода.

Список 183: Пример отрисовки в растре и сохранение результата в файле.

```
image = QImage(100, 100, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
point = Point(50, 50)
style = PointStyle.create_mi_font(42, Qt.red, 24)
style.draw(point, painter)
image.save(filename)
```

find_style(geom: [Geometry](#)) → [Style](#)

Возвращает стиль, подходящий для переданной геометрии.

classmethod for_geometry(geom: [Geometry](#)) → [Style](#)

Возвращает стиль по умолчанию из настроек для переданного объекта.

Параметры

geom - Геометрический объект, для которого необходимо получить соответствующий ему стиль.

classmethod for_geometry_type(geometry_type: [StyleGeometryType](#)) → [Style](#)

Возвращает стиль по умолчанию для переданного типа геометрии.

Параметры

geometry_type - Тип геометрического объекта, для которого необходимо получить стиль.

for_line(style: [LineStyle](#))

Задание стиля для линейных объектов [LineStyle](#).

for_point(style: [PointStyle](#))

Задание стиля для точечных объектов [PointStyle](#).

for_polygon(style: [PolygonStyle](#))

Задание стиля для полигональных объектов [PolygonStyle](#).

for_text(style: [TextStyle](#))

Задание стиля для текстовых объектов [TextStyle](#).

classmethod from_mapinfo(mapbasic_string: [str](#)) → [Optional\[Style\]](#)

Получает стиль из строки формата MapBasic.

Параметры

mapbasic_string - Строка в формате MapBasic.

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

property line: [Optional\[LineStyle\]](#)

Стиль для линейных объектов [LineStyle](#).

property point: `Optional[PointStyle]`

Стиль для точечных объектов `PointStyle`.

property polygon: `Optional[PolygonStyle]`

Стиль для полигональных объектов `PolygonStyle`.

property text: `Optional[TextStyle]`

Стиль для текстовых объектов `TextStyle`.

to_mapinfo() → `str`

Возвращает строковое представление в формате MapBasic.

```
print(style.to_mapinfo())
'''
>>> Pen (1, 2, 0) Brush (8, 255)
'''
```

22.14 TypeSqlDialect - Диалект при выполнении запросов

class `axipy.TypeSqlDialect`

Используемый диалект при выполнении SQL запросов. Есть отличия как в скорости, так и в функционале. выбор зависит от конкретной задачи. Значение, установленное по умолчанию можно получить посредством свойства `data_manager.sql_dialect`

Таблица 28: Значения

Значение	Наименование
<code>axioma</code>	Использовать собственную реализацию инструмента выполнения запросов
<code>sqlite</code>	Использовать при выполнении реализацию, являющийся надстройкой над <code>sqlite</code>

22.15 Raster - Растр

class `axipy.Raster`

Базовые классы: `DataObject`

Растровый объект.

Свойства:

<code>coordsystem</code>	Система координат раstra.
<code>device_to_scene_transform</code>	Матрица преобразования из точек на изображении (пиксели) в точки на карте.
<code>file_name</code>	Возвращает путь к файлу таблицы.
<code>is_spatial</code>	Признак того, что объект данных является пространственным.
<code>is_valid</code>	Возвращает признак валидности объекта.
<code>name</code>	Название объекта данных.
<code>properties</code>	Дополнительные свойства объекта данных.
<code>provider</code>	Провайдер изначального источника данных.
<code>scene_to_device_transform</code>	Матрица преобразования из точек на карте в точки на изображении (пиксели).
<code>size</code>	Размер раstra в пикселях.

Методы:

<code>close()</code>	Пытается закрыть таблицу.
<code>get_gcps()</code>	Возвращает точки привязки.

Сигналы:

<code>destroyed</code>	Сигнал оповещения об удалении объекта.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__()` → `DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object)` → `bool`

Оператор равенства (`==`).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType])` → `Optional`[`bool`]

Закрывает таблицу при выходе из блока кода `with`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`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 coordsystem: `Optional[CoordSystem]`

Система координат растра.

property destroyed: `Signal`

Сигнал оповещения об удалении объекта.

property device_to_scene_transform: `QTransform`

Матрица преобразования из точек на изображении (пиксели) в точки на карте.

property file_name: `Optional[Path]`

Возвращает путь к файлу таблицы.

get_gcps() → `List[GCP]`

Возвращает точки привязки.

property is_spatial: `bool`

Признак того, что объект данных является пространственным.

property is_valid: `bool`

Возвращает признак валидности объекта.

property name: `str`

Название объекта данных.

property properties: `dict`

Дополнительные свойства объекта данных.

property provider: `str`

Провайдер изначального источника данных.

property scene_to_device_transform: `QTransform`

Матрица преобразования из точек на карте в точки на изображении (пиксели).

property size: `QSize`

Размер растра в пикселях.

22.16 raster - Операції с растром

Модуль операцій с растрами.

class ахіру.GCP

Точка привязки (Ground Control Point).

Атрибути:

device	Точка на зображенні (пиксели).
label	Ідентифікатор.
scene	Точка на карті.

Спеціальні методи:

__repr__()	Офіційне рядкове представлення об'єкта при виклику <code>repr()</code> .

class ахіру.Algorithm

Алгоритм трансформації.

Атрибути:

POLYNOM1	Афінитет.
POLYNOM2	Полиноміальний другого порядку.
POLYNOM3	Полиноміальний третього порядку.
SPLINE	Сплайновий.

class ахіру.Resample

Метод інтерполяції.

Атрибути:

Average	Середній.
Bilinear	Білінійний.
Cubic	Кубічний.
CubicSpline	Кубічний сплайн.
Lanczos	Ланцоша.
Max	Максимальний.
Med	Медіанний.
Min	Мінімальний.
Mode	Самий зустрічаючийся.
NearestNeighbour	Ближайший.
Q1	Перший кватиль.
Q3	Третій кватиль.
Sum	Сума.

class ахіру.Format

Формат зображення.

Атрибути:

BMP

GTiff

JPEG

PNG

class axipy.Compression

Сжатие.

Примечание: Сжатие можно использовать только для файлов формата GeoTIFF.

Атрибуты:

DEFLATE

LZW

NONE

PACKBITS

axipy.**register**(filepath: str, bindings: Union[List[GCP], QTransform], coordsystem: CoordSystem, override: bool = False)

Регистрирует растр.

Добавляет изображению пространственную привязку.

Параметры

- **filepath** – Файл с изображением.
- **bindings** – Привязка в виде точек или матрицы преобразования.
- **coordsystem** – Координатная система.
- **override** – При регистрации формируется файл привязки TAB. Если он существует, то данный параметр управляет его перезаписью.

Список 184: Пример использования

```
from PySide2.QtGui import QTransform

matrix = QTransform()
coordsystem = CoordSystem.from_units(Unit.m)
register(imagefile, matrix, coordsystem)
```

axipy.**transform**(inputfile: str, outputfile: str, points: List[GCP], coordsystem: CoordSystem, algorithm: Algorithm = Algorithm.SPLINE, resample: Resample = Resample.NearestNeighbour, output_format: Format = Format.GTiff, compression: Compression = Compression.NONE)

Трансформирует растр.

Растру, имеющему пространственную привязку, задает новую привязку. На выходе получается новый растр.

Параметры

- **inputfile** – Входной файл с растром.
- **outputfile** – Выходной файл с растром.
- **points** – Точки привязки.
- **coordsystem** – Координатная система.
- **algorithm** – Алгоритм трансформации.
- **resample** – Метод интерполяции.
- **output_format** – Выходной формат.
- **compression** – Метод сжатия.

Список 185: Пример использования

```
coordsystem = CoordSystem.from_epsg(4326)
gcps = [
    GCP((0, 0), (0, 0)),
    GCP((200, 0), (30, 30)),
    GCP((200, 200), (60, 0)),
]
transform(rasterfile, outputfile, gcps, coordsystem)
```

22.17 ObserverManager - Менеджер наблюдателей

class axipy.ObserverManager

Наблюдатели за состоянием. Класс является словарем, доступным только для чтения (`collections.abc.Mapping`), где ключи это имена наблюдателей, а значения это объекты класса `axipy.Observer`. Поддерживает обращение по ключу.

Примечание: Создание `axipy.ObserverManager` не требуется, используйте объект `axipy.observer_manager`.

Атрибуты:

<code>ActiveMapView</code>	Есть активное окно карты.
<code>ActiveTableView</code>	Есть активное окно таблицы.
<code>ActiveView</code>	Есть активное окно.
<code>Editable</code>	Активная карта имеет редактируемый слой.
<code>HasTables</code>	Открыта хотя бы одна таблица.
<code>Selection</code>	Есть выборка.
<code>SelectionEditable</code>	Карта имеет редактируемый слой и есть выделенные объекты на одном из слоев карты.
<code>SelectionEditableIsSame</code>	Карта имеет редактируемый слой и выборку на этом слое.

Методы:

<code>get(key[, default_value])</code>	Возвращает значение по ключу.
<code>items()</code>	Возвращает набор кортежей ключ-значение, где ключи это имена наблюдателей, а значения это объекты класса <code>axipy.Observer</code> .
<code>keys()</code>	Возвращает набор ключей, где ключи это имена наблюдателей.
<code>remove(name)</code>	Удаляет наблюдатель по имени.
<code>values()</code>	Возвращает коллекцию значений, где значения это объекты класса <code>axipy.Observer</code> .

Специальные методы:

<code>__contains__(key)</code>	Проверяет наличие элемента в объекте (оператор <code>in</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__getitem__(key)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

ActiveMapView: `Observer`

Есть активное окно карты.

ActiveTableView: `Observer`

Есть активное окно таблицы.

ActiveView: `Observer`

Есть активное окно.

Editable: `Observer`

Активная карта имеет редактируемый слой.

HasTables: [Observer](#)

Открыта хотя бы одна таблица.

Selection: [Observer](#)

Есть выборка.

SelectionEditable: [Observer](#)

Карта имеет редактируемый слой и есть выделенные объекты на одном из слоев карты.

SelectionEditableIsSame: [Observer](#)

Карта имеет редактируемый слой и выборку на этом слое.

`__contains__`(key)

Проверяет наличие элемента в объекте (оператор `in`).

`__eq__`(other)

Оператор равенства (`==`).

`__getitem__`(key: [str](#)) → [Observer](#)

Доступ к элементу по индексу (`obj[i]`).

`__iter__`() → [Iterator](#)[[str](#)]

Возвращает итератор по объекту (функция `iter()`).

`__len__`() → [int](#)

Длина объекта (функция `len()`).

`__repr__`() → [str](#)

Официальное строковое представление объекта при вызове `repr()`.

`get`(key: [str](#), default_value: [Optional](#)[[Any](#)] = None) → [Optional](#)[[Observer](#)]

Возвращает значение по ключу.

`items`() → [ItemsView](#)[[str](#), [Observer](#)]

Возвращает набор кортежей ключ-значение, где ключи это имена наблюдателей, а значения это объекты класса [axipy.Observer](#).

`keys`() → [KeysView](#)[[str](#)]

Возвращает набор ключей, где ключи это имена наблюдателей.

`remove`(name: [str](#))

Удаляет наблюдатель по имени.

Параметры

`name` – Имя наблюдателя.

`values`() → [ValuesView](#)[[Observer](#)]

Возвращает коллекцию значений, где значения это объекты класса [axipy.Observer](#).

22.18 Observer - Наблюдатель

class axipy.Observer

Наблюдатель.

При создании, наблюдатель автоматически добавляется в менеджер наблюдателей `axipy.observer_manager`. Чтобы удалить наблюдатель, используйте метод `axipy.ObserverManager.remove`.

Свойства:

<code>name</code>	Возвращает имя наблюдателя.
<code>value</code>	Устанавливает или возвращает значение наблюдателя.

Сигналы:

<code>changed</code>	Сигнал об изменении значения.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(name: str, init_value: Any)`

Конструктор класса.

Параметры

- **name** - Имя наблюдателя.
- **init_value** - Начальное значение наблюдателя.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property `changed: Signal`

Сигнал об изменении значения.

Тип результата

`Signal[Any]`

```
import axipy

def print_func(value):
    print(value)

axipy.observer_manager.Selection.changed.connect(print_func)
```

property `name: str`

Возвращает имя наблюдателя.

property value: *Any*

Устанавливает или возвращает значение наблюдателя.

При изменении значения испускается сигнал *changed*.

22.19 TabFile - Файл TAB

class *axipy.TabFile*

Класс поддержки файла TAB формата MapInfo.

Методы:

<code>generate_tab(data_object, out_file[, ...])</code>	Генерирует файл TAB для переданного открытого объекта, если такую возможность поддерживает провайдер данных.
<code>suggest_tab_name(data_object)</code>	Сервисная функция.

generate_tab(data_object: *DataObject*, out_file: *str*, override: *bool* = True, linked_file: *bool* = True) → *bool*

Генерирует файл TAB для переданного открытого объекта, если такую возможность поддерживает провайдер данных.

Параметры

- **data_object** – открытый объект данных, для которого необходимо создать файл TAB.
- **out_file** – Имя файла с расширением *tab*. Как вариант, можно использовать результат `suggest_tab_name()`.
- **override** – Перезаписывать файл. Если установлено *False* и файл существует, будет выброшено исключение *FileExistsError*
- **linked_file** – Если файл генерируется для СУБД, и при установке значения *True* будет создан связанный файл

Результат

Возвращает *True*, если успешно.

Создание TAB файла для открытой таблицы или раstra:

```
filepath = 'world.tif'
out_file_name = 'world.tab'
tab = TabFile()
tab.generate_tab(table, out_file_name)
```

Создание TAB файла для открытого источника тайлового сервиса:

```
prj_mercator = (
    'Earth Projection 10, 104, "m", 0 '
    'Bounds (-20037508.34, -20037508.34) (20037508.34, 20037508.34)'
)
osm_raster = axipy.provider_manager.tms.open(
    'http://maps.axioma-gis.ru/osm/{LEVEL}/{ROW}/{COL}.png',
```

(continues on next page)

(продолжение с предыдущей страницы)

```
prj=prj_mercator
)
tab = axipy.TabFile()
out_file_name = tab.suggest_tab_name(osm_raster)
tab.generate_tab(osm_raster, out_file_name)
```

Загрузка данных из БД PostgreSQL в виде связанной таблицы:

```
definition = provider_manager.postgre.get_source(
    host='localhost',
    db_name='test',
    user='test',
    password='pass',
    dataobject='world'
)
table = provider_manager.open(definition)
out_file_name = tab.suggest_tab_name(table)
tab = TabFile()
tab.generate_tab(table, out_file_name, linked_file = True)
```

suggest_tab_name(data_object: [DataObject](#)) → [str](#)

Сервисная функция.

Предлагает наименование TAB файла для объекта данных. Результат можно использовать в методе [generate_tab\(\)](#) в качестве имени выходного файла.

22.20 RasteredTable - Источники ГИС Панорама и AutoCAD.

class axipy.**RasteredTable**

Базовые классы: [DataObject](#)

Данные из таких источников, как ГИС Панорама и AutoCAD.

Свойства:

coordsystem	Система координат.
file_name	Возвращает путь к файлу таблицы.
is_spatial	Признак того, что объект данных является пространственным.
is_valid	Возвращает признак валидности объекта.
layers	Список доступных для запроса данных слоев.
name	Название объекта данных.
properties	Дополнительные свойства объекта данных.
provider	Провайдер изначального источника данных.
schema	Схема таблицы.

Методы:

<code>close()</code>	Пытается закрыть таблицу.
<code>items([layer_name])</code>	Запрашивает записи из источника.

Сигналы:

<code>destroyed</code>	Сигнал оповещения об удалении объекта.

Специальные методы:

<code>__enter__()</code>	Вход в контекст (используется с оператором <code>with</code>).
<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__exit__(exception_type, exception_value, ...)</code>	Закрывает таблицу при выходе из блока кода <code>with</code> .
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__enter__() → DataObject`

Вход в контекст (используется с оператором `with`).

`__eq__(other: object) → bool`

Оператор равенства (`==`).

`__exit__(exception_type: Optional[Type[BaseException]], exception_value: Optional[BaseException], traceback: Optional[TracebackType]) → Optional[bool]`

Закрывает таблицу при выходе из блока кода `with`.

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`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 `coordsystem: Optional[CoordSystem]`

Система координат.

property `destroyed: Signal`

Сигнал оповещения об удалении объекта.

property `file_name: Optional[Path]`

Возвращает путь к файлу таблицы.

property is_spatial: bool

Признак того, что объект данных является пространственным.

property is_valid: bool

Возвращает признак валидности объекта.

items(layer_name: Optional[str] = None) → Iterator[Feature]

Запрашивает записи из источника.

Параметры

layer_name - Наименование слоя, по которому будет произведена выборка. Если значение пустое, выдаются данные по всем слоям.

Результат

Итератор по записям.

property layers: List[str]

Список доступных для запроса данных слоев.

property name: str

Название объекта данных.

property properties: dict

Дополнительные свойства объекта данных.

property provider: str

Провайдер изначального источника данных.

property schema: Schema

Схема таблицы.

22.21 GeometryClass - Класс геометрических объектов

class axipy.GeometryClass

Класс геометрических объектов Table.classGeometries.

Атрибуты:

Unknown	Тип неопределен.
Points	Точечные объекты.
Lines	Линейные объекты.
Polygons	Площадные объекты.
Texts	Текстовые объекты.

axipy.render - Модуль отрисовки.

Модуль отрисовки.

Данный модуль содержит инструменты, предназначенные для отрисовки геопространственных и прочих данных.

23.1 Map - Карта

`class axipy.Map`

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 расстояний `distanceUnit` и площадей `areaUnit` берутся из настроек по умолчанию.

Параметры

- **layers** – Список слоев, с которым будет создана карта.
- **preserve_order** – Порядок слоев оставить как есть, в противном случае будет произведена попытка отсортировать слои в соответствии с контентом (растры вниз, точечные объекты вверх)

Исключение

ValueError – Если один и тот же слой был передан несколько раз.

Список 1: Пример.

```
table_world = provider_manager.openfile(filepath)
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)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
'''
>>> CK: Earth Projection 12, 62, "m", 0
>>> Охват: (-16194966.287183324 -8621185.324024437) (16789976.633236416 8326222.
↳ 646170927)
>>> Единицы измерения расстояний: километры
>>> Единицы измерения расстояний (изменено): мили
'''
```

Свойства:

areaUnit	Устанавливает или возвращает единицы измерения площадей карты.
cosmetic	Возвращает косметический слой карты.
custom_labels	Возвращает пользовательские метки.
distanceUnit	Устанавливает или возвращает единицы измерения расстояний на карте.
editable_layer	Устанавливает или возвращает редактируемый слой для текущей карты.
layers	Возвращает список слоев и групп слоев.

Методы:

draw(context)	Рисует карту в контексте.
draw_vector(context)	Рисует карту в контексте в виде вектора.
get_best_coordsystem()	Определяет координатную системы карты, наиболее подходящую исходя из содержимого перечня слоев.
get_best_rect([coordsystem])	Определяет ограничивающий прямоугольник карты.
to_image(width, height[, coordsystem, bbox])	Рисует карту в изображение.

Сигналы:

need_redraw	Сигнал о необходимости перерисовки карты.

Специальные методы:

__contains__(layer)	Проверяет наличие элемента в объекте (оператор in).
__eq__(other)	Оператор равенства (==).
__str__()	Неформальное строковое представление объекта при вызове str или print().

__contains__(layer: [Layer](#)) → bool

Проверяет наличие элемента в объекте (оператор in).

__eq__(other: [object](#)) → bool

Оператор равенства (==).

__str__() → str

Неформальное строковое представление объекта при вызове `str` или `print()`.

property areaUnit: [AreaUnit](#)

Устанавливает или возвращает единицы измерения площадей карты.

property cosmetic: [CosmeticLayer](#)

Возвращает косметический слой карты.

property custom_labels: [CustomLabels](#)

Возвращает пользовательские метки.

Список 2: Пример.

```
table_world = provider_manager.openfile(filepath)
world_layer = Layer.create(table_world)
map_ = Map([world_layer])
layer = map_.layers[0]
# Определим свойства
p = CustomLabelProperties()
# Переназначим выражение
p.expression = 'Новое выражение'
# Угол поворота
p.angle = 30
# Выноска будет в виде стрелки
p.endType = CustomLabelEndType.Arrow
# Положение выноски
p.position = Point(100, 200)
# Выравнивание относительно этого положения
p.position_align = LabelLayoutPosition.TopRight
# Устанавливаем стиль выноски
p.arrow_style = Style.from_mapinfo('Pen (2, 86, 16711680)')
# Устанавливаем стиль текста для метки
text_style = TextStyle("Droid Sans", 12)
text_style.color = Qt.red
text_style.bold = True
p.text_style = text_style
# Установим свойства
map_.custom_labels.set(layer, 1, p)
# Запрос свойств
props = map_.custom_labels.get(layer, 1)
# Перечень идентификаторов объектов с метками
ids = map_.custom_labels.ids(layer)
# Установка параметров по умолчанию для новых меток
s = Style.from_mapinfo('Pen (2, 86, 16711680)')
map_.custom_labels.set_default_arrow_type(layer, s)
map_.custom_labels.set_default_end_type(layer, CustomLabelEndType.Line)
```

property distanceUnit: [LinearUnit](#)

Устанавливает или возвращает единицы измерения расстояний на карте.

draw(context: [Context](#))

Рисует карту в контексте.

Параметры

context – Контекст рисования.

Список 3: Пример получения карты как растра.

```
in_filepath = 'world.tab'
table_world = provider_manager.openfile(in_filepath)
m = Map([table_world])
image = QImage(1600, 1000, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.transparent)
painter = QPainter(image)
context = Context(painter)
m.draw(context)
out_filepath = 'world.png'
image.save(out_filepath)
```

draw_vector(context: [Context](#))

Рисует карту в контексте в виде вектора. Это может потребоваться при генерации карты в виде, к примеру, SVG.

Параметры

context – Контекст рисования.

Список 4: Пример получения карты в виде вектора.

```
in_filepath = 'world.tab'
out_filepath = 'world.svg'
table_world = provider_manager.openfile(in_filepath)
m = Map([table_world])
generator = QSvgGenerator()
generator.setFileName(out_filepath)
generator.setSize(QSize(1600,1000));
painter_svg = QPainter(generator)
context = Context(painter_svg)
m.draw_vector(context)
```

property editable_layer: [Optional\[VectorLayer\]](#)

Устанавливает или возвращает редактируемый слой для текущей карты.

Исключение

ValueError – При попытке установить слой, не принадлежащий этой карте.

get_best_coordsystem() → [CoordSystem](#)

Определяет координатную системы карты, наиболее подходящую исходя из содержимого перечня слоев.

get_best_rect(coordsystem: [Optional\[CoordSystem\]](#) = None) → [Rect](#)

Определяет ограничивающий прямоугольник карты.

Параметры

coordsystem – Координатная система, в которой необходимо получить результат. Если отсутствует, будет выдан результат для наиболее подходящей координатной системы.

property layers: ListLayers

Возвращает список слоев и групп слоев.

Примечание: Не содержит косметический слой `cosmetic`.

Список 5: Примеры доступа.

```
# Создадим карту с тремя слоями
map = Map([world, worldcap, russia])
print(len(map.layers))
...

>>> 3
...

print(map.layers[0].title)
...

>>> world
...

# Группировка первый двух слоев с именем "Мир"
map.layers.group([0, 1], 'Мир')
# Перечень элементов
for l in map.layers:
    if isinstance(l, Layer):
        print('Слой:', l.title)
    elif isinstance(l, ListLayers):
        print('Группа:', l.title)
...

>>> Группа: Мир
>>> Слой: russia
...

# Изменение позиции
map.layers.move(0, 1)
# Управление видимостью группы слоев
map.layers[1].visible = False
# Разгруппировка ранее созданной группы
map.layers.ungroup(1)
# Удаление слоя из карты
map.layers.remove(1)
# Добавление пустой группы
map.layers.add_group('Новая группа')
```

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.

Тип результата
Signal[]

Список 6: Пример.

```
# Смотрим активное окно.
if isinstance(view_manager.active, MapView):
    # Если это карта, подключимся к событию обновления окна этой карты.
    map_view = view_manager.active
    map_view.map.need_redraw.connect(lambda: print('Update map'))
```

`to_image`(width: `int`, height: `int`, coordsystem: `Optional[CoordSystem]` = None, bbox: `Optional[Rect]` = None) → `QImage`

Рисует карту в изображение.

Параметры

- **width** – Ширина выходного изображения.
- **height** – Высота выходного изображения.
- **coordsystem** – Координатная система. Если не задана, берется наиболее подходящая.
- **bbox** – Ограничивающий прямоугольник. Если не задан, берется у карты.

Результат

Изображение.

23.2 ListLayers - Список слоев карты

`class axipy.ListLayers`

Группа слоев.

Может включать в себя как слои `axipy.Layer` так и группы слоев `axipy.ListLayers`.
Пример использования см `axipy.Map.layers`

Свойства:

<code>count</code>	Возвращает количество слоев и групп слоев.
<code>title</code>	Устанавливает или возвращает наименование группы.
<code>visible</code>	Устанавливает или возвращает признак видимости группы.

Методы:

<code>add_group(name)</code>	Создает пустую группу.
<code>append(layer)</code>	Добавляет слой в карту.
<code>at(index)</code>	Возвращает слой или группы слоев по их индексу.
<code>group(indexes, name)</code>	Группировка слоев и групп в соответствие со списком их индексов.
<code>insert(layer)</code>	Добавляет слой в карту.
<code>move(from_index, to_index)</code>	Перемещает слой или вложенную группу слоев в списке слоев по его индексу.
<code>remove(index)</code>	Удаляет слой по индексу.
<code>ungroup(index)</code>	Разгруппировка группы слоев по его индексу.

Специальные методы:

<code>__getitem__(index)</code>	Возвращает слой или группу по их индексу.
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__getitem__(index: int) → Union[Layer, ListLayers]`

Возвращает слой или группу по их индексу.

Параметры

index – Индекс слоя или группы слоев.

Результат

Искомый элемент.

`__iter__() → Iterator[Union[Layer, ListLayers]]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

add_group(name: str)

Создает пустую группу.

Параметры

name – Наименование создаваемой группы.

append(layer: Layer)

Добавляет слой в карту. Добавление группы слоев не поддерживается и производится путем группировки существующих элементов посредством метода `group()`.

Параметры

layer – Добавляемый слой.

Исключение

ValueError – Если слой уже содержится в карте.

at(index: int) → Union[Layer, ListLayers]

Возвращает слой или группы слоев по их индексу.

Параметры

index – Индекс слоя или группы в списке.

Например:

```
layers.at(2)
layers[2]
```

property count: int

Возвращает количество слоев и групп слоев.

Так же допустимо использование функции `len()`.

group(indexes: List[int], name: str)

Группировка слоев и групп в соответствие со списком их индексов. При этом создается новая группа и все элементы (слои и группы слоев) помещаются внутрь этой группы.

Параметры

- **indexes** – Список индексов элементов, которые необходимо объединить.
- **name** – Наименование создаваемой группы.

insert(layer: **Layer**)

Добавляет слой в карту. В отличие от **ListLayers.append()** при вставке слоя производится попытка вставить его в список в зависимости от контента.

Параметры

layer – Вставляемый слой.

Исключение

ValueError – Если слой уже содержится в карте.

move(from_index: **int**, to_index: **int**)

Перемещает слой или вложенную группу слоев в списке слоев по его индексу.

Параметры

- **from_index** – Индекс слоя для перемещения.
- **to_index** – Целевой индекс.

remove(index: **int**)

Удаляет слой по индексу.

Параметры

index – Индекс удаляемого слоя.

property title: str

Устанавливает или возвращает наименование группы.

ungroup(index: **int**)

Разгруппировка группы слоев по его индексу. при этом все внутренние элементы переносятся на верхний уровень данного списка. Если по индексу располагается не группа, то будет выброшено исключение.

Параметры

index – Индекс группы слоев.

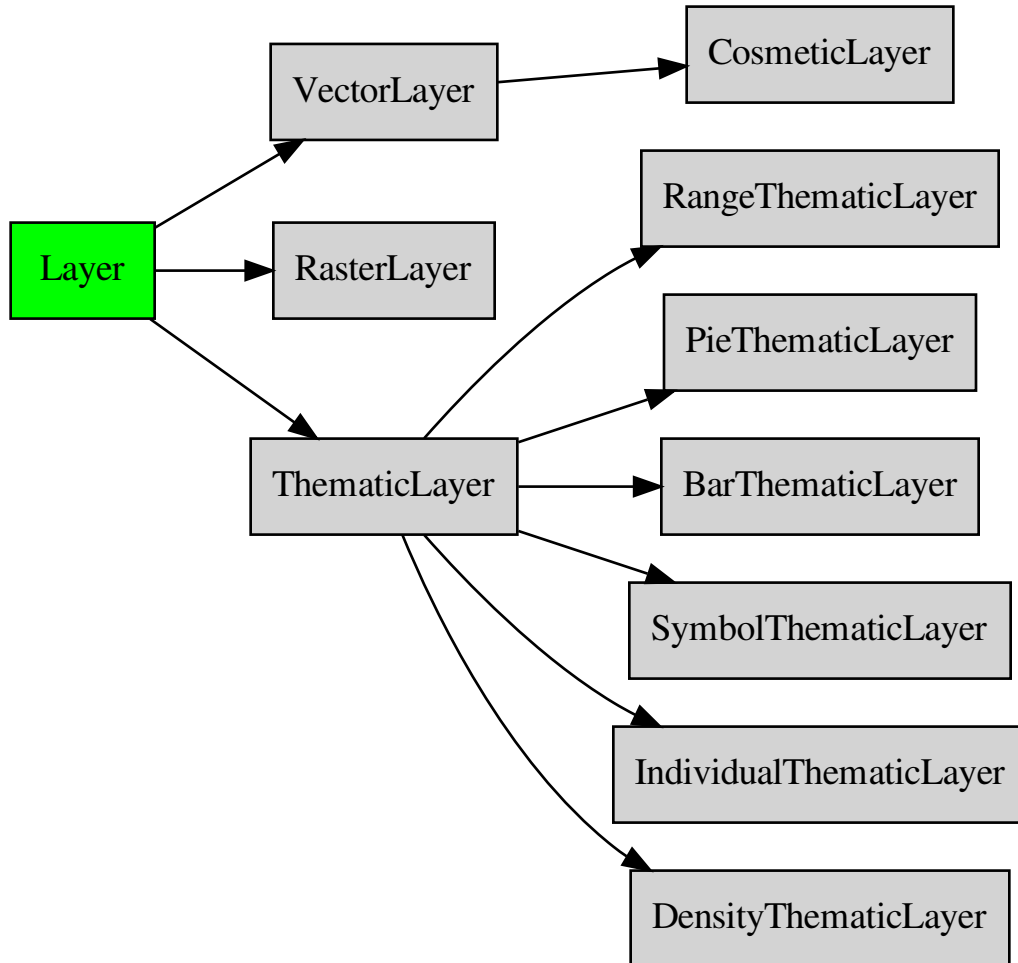
property visible: bool

Устанавливает или возвращает признак видимости группы.

23.3 Layer - Слой

23.3.1 Layer - Слой

Иерархия классов слоев карты:



class ахіру.Layer

Абстрактный базовый класс для слоя карты.

Для создания нового экземпляра для векторного или растрового источника данных необходимо использовать метод `Layer.create()`. Для тематических слоев — использовать соответствующие им конструкторы.

Классовые методы:

<code>create()</code>	Создает слой на базе открытой таблицы или раstra.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: Raster) → RasterLayer

classmethod create(dataObject: Table) → VectorLayer

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан `VectorLayer` или `RasterLayer`.

Список 7: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

`Signal[]`

property data_object: DataObject

Возвращает источник данных для слоя.

get_bounds() → Rect

Возвращает область, которая содержит все данные слоя.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном `zoom_restrict=True`.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном `zoom_restrict=True`.

property need_redraw: [Signal](#)

Сигнал о необходимости перерисовать слой.

Тип результата

[Signal\[\]](#)

property opacity: [int](#)

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property selectable: [bool](#)

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property title: [str](#)

Устанавливает или возвращает наименование слоя.

property visible: [bool](#)

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: [bool](#)

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств `zoom_min` и `zoom_max`.

23.3.2 RasterLayer - Растровый слой

class axipy.RasterLayer

Базовые классы: [Layer](#)

Класс, который должен использоваться в качестве базового класса для тех слоев, в которых используются свойства отрисовки растрового изображения.

Примечание: Создание слоя производится посредством метода вызова [Layer.create\(\)](#).

Список 8: Примеры создания растрового слоя.

```
raster = provider_manager.openfile(filename)
raster_layer = Layer.create(raster)
raster_layer.transparentColor = QColor('#000014')
```

Классовые методы:

[create\(dataObject\)](#)

Создает слой на базе открытой таблицы или растра.

Свойства:

brightness	Устанавливает или возвращает значение яркости.
contrast	Устанавливает или возвращает значение контраста.
coordsystem	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
data_object	Возвращает источник данных для слоя.
grayscale	Устанавливает или возвращает признак, является ли данное изображение черно-белым.
is_valid	Возвращает признак валидности слоя.
max_zoom	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
min_zoom	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
opacity	Устанавливает или возвращает прозрачность слоя в составе карты.
selectable	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
title	Устанавливает или возвращает наименование слоя.
transparentColor	Устанавливает или возвращает цвет раstra, который обрабатывается как прозрачный.
visible	Устанавливает или возвращает признак видимости слоя.
zoom_restrict	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

get_bounds()	Возвращает область, которая содержит все данные слоя.

Сигналы:

data_changed	Сигнал об изменении контента слоя.
need_redraw	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (`==`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property brightness: int

Устанавливает или возвращает значение яркости.

Значение может быть в пределах от -100 до 100.

property contrast: int

Устанавливает или возвращает значение контраста.

Значение может быть в пределах от -100 до 100.

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан `VectorLayer` или `RasterLayer`.

Список 9: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

`Signal[]`

property data_object: DataObject

Возвращает источник данных для слоя.

get_bounds() → Rect

Возвращает область, которая содержит все данные слоя.

property grayscale: bool

Устанавливает или возвращает признак, является ли данное изображение черно-белым.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property title: str

Устанавливает или возвращает наименование слоя.

property transparentColor: QColor

Устанавливает или возвращает цвет растра, который обрабатывается как прозрачный.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.3.3 VectorLayer - Векторный слой

class axipy.VectorLayer

Базовые классы: Layer

Слой, основанный на базе векторных данных.

Примечание: Создание слоя производится посредством метода вызова `Layer.create()`

Список 10: Примеры работы со свойствами слоя.

```
# Зададим в качестве формулы метки атрибут "Страна" и запретим перекрытие меток
↪ друг другом:
world.label.text = "Страна"
world.label.placementPolicy = LabelOverlap.DisallowOverlap
# Задание стиля оформления слоя
style_layer = Style.from_mapinfo("Pen (1, 2, 0) Brush (8, 255) Symbol (33,255,14)")
world.overrideStyle = style_layer
# Для сброса переопределения достаточно задать значение None
world.overrideStyle = None
```

Классовые методы:

`create(dataObject)`

Создает слой на базе открытой таблицы или растра.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>hotlink</code>	Устанавливает или возвращает наименование атрибута таблицы для хранения гиперссылки.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>label</code>	Возвращает метки слоя.
<code>linesDirectionVisibile</code>	Устанавливает или возвращает признак отображения направлений линий.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>nodesVisible</code>	Устанавливает или возвращает признак отображения узлов линий и полигонов.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>overrideStyle</code>	Устанавливает или возвращает переопределяемый стиль слоя.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>showCentroid</code>	Устанавливает или возвращает признак отображения центроидов на слое.
<code>thematic</code>	Возвращает перечень тематик для данного слоя.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property coordsystem: **CoordSystem**

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod **create**(dataObject: **DataObject**) → **Layer**

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан **VectorLayer** или **RasterLayer**.

Список 11: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: **Signal**

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: **Table**

Возвращает источник данных для слоя.

get_bounds() → **Rect**

Возвращает область, которая содержит все данные слоя.

property hotlink: **str**

Устанавливает или возвращает наименование атрибута таблицы для хранения гиперссылки.

Таблица 1: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

property is_valid: `bool`

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property label: `Label`

Возвращает метки слоя.

В качестве формулы может использоваться или наименование поля таблицы или выражение.

property linesDirectionVisibile: `bool`

Устанавливает или возвращает признак отображения направлений линий.

property max_zoom: `float`

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: `float`

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: `Signal`

Сигнал о необходимости перерисовать слой.

Тип результата
`Signal[]`

property nodesVisible: `bool`

Устанавливает или возвращает признак отображения узлов линий и полигонов.

property opacity: `int`

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property overrideStyle: `Optional[Style]`

Устанавливает или возвращает переопределяемый стиль слоя.

Если задан как None (по умолчанию), объекты будут отображены на основании оформления источника данных.

property selectable: `bool`

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property showCentroid: bool

Устанавливает или возвращает признак отображения центроидов на слое.

property thematic: ListThematic

Возвращает перечень тематик для данного слоя. Работа с тематическими слоями похожа на работу со списком list.

Список 12: Пример.

```
# Создадим тематический слой
rangel = RangeThematicLayer("Население")
# Добавим в основной слой
world.thematic.append(rangel)
# Получим добавленный тематический слой
rangel = world.thematic[0]
# Просмотр всех тематик слоя
for t in world.thematic:
    print('thematic:', t.title)
```

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.3.4 CosmeticLayer - Косметический слой

class axipy.CosmeticLayer

Базовые классы: [VectorLayer](#)

Косметический слой.

Классовые методы:

<code>create(dataObject)</code>	Создает слой на базе открытой таблицы или раstra.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>hotlink</code>	Устанавливает или возвращает наименование атрибута таблицы для хранения гиперссылки.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>label</code>	Возвращает метки слоя.
<code>linesDirectionVisibile</code>	Устанавливает или возвращает признак отображения направлений линий.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>nodesVisible</code>	Устанавливает или возвращает признак отображения узлов линий и полигонов.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>overrideStyle</code>	Устанавливает или возвращает переопределяемый стиль слоя.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>showCentroid</code>	Устанавливает или возвращает признак отображения центроидов на слое.
<code>thematic</code>	Возвращает перечень тематик для данного слоя.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property coordsystem: **CoordSystem**

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod **create**(dataObject: **DataObject**) → **Layer**

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан **VectorLayer** или **RasterLayer**.

Список 13: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: **Signal**

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: **Table**

Возвращает источник данных для слоя.

get_bounds() → **Rect**

Возвращает область, которая содержит все данные слоя.

property hotlink: **str**

Устанавливает или возвращает наименование атрибута таблицы для хранения гиперссылки.

Таблица 2: Возможны следующие варианты

Значение	Описание
axioma://world.tab	Открывает файл или рабочее пространство в аксиоме
addlayer://world	Добавляет слой world в текущую карту
exec://gimp	Запускает на выполнение программу gimp
https://axioma-gis.ru/	Открывает ссылку в браузере

Если префикс отсутствует, то производится попытка запустить по ассоциации.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property label: Label

Возвращает метки слоя.

В качестве формулы может использоваться или наименование поля таблицы или выражение.

property linesDirectionVisibile: bool

Устанавливает или возвращает признак отображения направлений линий.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата
Signal[]

property nodesVisible: bool

Устанавливает или возвращает признак отображения узлов линий и полигонов.

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property overrideStyle: Optional[Style]

Устанавливает или возвращает переопределяемый стиль слоя.

Если задан как None (по умолчанию), объекты будут отображены на основании оформления источника данных.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property showCentroid: bool

Устанавливает или возвращает признак отображения центроидов на слое.

property thematic: ListThematic

Возвращает перечень тематик для данного слоя. Работа с тематическими слоями похожа на работу со списком list.

Список 14: Пример.

```
# Создадим тематический слой
rangel = RangeThematicLayer("Население")
# Добавим в основной слой
world.thematic.append(rangel)
# Получим добавленный тематический слой
rangel = world.thematic[0]
# Просмотр всех тематик слоя
for t in world.thematic:
    print('thematic:', t.title)
```

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.3.5 ListThematic - Перечень тематик для векторного слоя

class axipy.ListThematic

Список тематических слоев (тематик) карты.

Свойства:

<code>count</code>	Возвращает количество тематик слоя.

Методы:

<code>append(lay)</code>	Добавить тематику.
<code>at(idx)</code>	Получение тематики по ее индексу.
<code>move(fromIdx, toIdx)</code>	Поменять тематики местами.
<code>remove(idx)</code>	Удалить тематику.

Специальные методы:

<code>__getitem__(item)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__getitem__(item: Union[int, str]) → Optional[ThematicLayer]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[ThematicLayer]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`append(layer: ThematicLayer)`

Добавить тематику.

Параметры

layer – Добавляемый тематический слой.

`at(idx: int) → Optional[ThematicLayer]`

Получение тематики по ее индексу.

Параметры

idx – Индекс запрашиваемой тематики.

`property count: int`

Возвращает количество тематик слоя.

`move(fromIdx: int, toIdx: int)`

Поменять тематики местами.

Параметры

- **fromIdx** – Текущий индекс.
- **toIdx** – Новое положение.

`remove(idx: int)`

Удалить тематику.

Параметры

idx – Индекс удаляемого слоя.

23.3.6 Метки

Label - Метка для векторного слоя

class ахіру.Label

Метки слоя. Доступны через свойство векторного слоя `VectorLayer.label`.

Список 15: Пример использования.

```
# Открываем таблицу
table = provider_manager.openfile(filepath)
# Создаем слой
layer = Layer.create(table)
# Формула метки
layer.label.text = 'Страна'
# Видимость
layer.label.visible = True
# Если метки перекрывают друг друга, ищем другое положение
layer.label.placementPolicy = LabelOverlap.OtherPosition
# Цвет шрифта
layer.label.color = Qt.blue
# устанавливаем прозрачность
layer.label.opacity = 50
# Показываем в пределах (0...3000км)
layer.label.rangeEnabled = True
layer.label.rangeMax = 3000000
# Положение подписей для точечных объектов
p_layout = layer.label.pointLayout
p_layout.position = LabelLayoutPosition.BottomRight
p_layout.visible = True
p_layout.offset = QSize(3, 3)
layer.label.pointLayout = p_layout
# Положение подписей для линейных объектов
l_layout = layer.label.lineLayout
l_layout.position = LabelLayoutPosition.Bottom
layer.label.lineLayout = l_layout
# Горизонтальное выравнивание подписей для линий
layer.label.horizontalAlign = LabelHorizontalAlign.Center
# Игнорируем дубликаты
layer.label.supressDuplicates = True
# Свес линии
layer.label.overhang = 67
```

Свойства:

<code>areaInterior</code>	Режим подписей для областей.
<code>areaLayout</code>	Положение подписей для областей.
<code>areaPosition</code>	Режим подписей для областей.
<code>backgroundColor</code>	Цвет фона.
<code>backgroundSize</code>	Толщина фона в пунктах.
<code>backgroundType</code>	Фон подписи.
<code>color</code>	Цвет шрифта меток.
<code>font</code>	Шрифт.
<code>horizontalAlign</code>	Горизонтальное выравнивание подписей.
<code>lineKeepDirection</code>	Направление текста строится вдоль направления линии.
<code>lineLayout</code>	Положение подписей для линий.
<code>linePosition</code>	Режим подписей для линий.
<code>opacity</code>	Прозрачность (0..100).
<code>overhang</code>	Максимальный свес для линии (в %).
<code>placementPolicy</code>	Принцип наложения меток на слой карты.
<code>pointLayout</code>	Положение подписей для точек.
<code>rangeEnabled</code>	Показывать в пределах.
<code>rangeMax</code>	Максимальный предел показа с метрах при включенном свойстве <code>rangeEnabled</code> .
<code>rangeMin</code>	Минимальный предел показа с метрах при включенном свойстве <code>rangeEnabled</code> .
<code>shadow</code>	Тень.
<code>spacing</code>	Разрядка.
<code>supressDuplicates</code>	Запретить повтор подписей.
<code>text</code>	Наименование атрибута таблицы либо выражение для метки, которое может основываться на одном или нескольких атрибутах.
<code>useClip</code>	Использовать динамические подписи.
<code>visible</code>	Управляет видимостью меток.

property areaInterior: LabelAreaInterior

Режим подписей для областей.

По умолчанию LabelAreaInterior.Centroid.

property areaLayout: LabelLayout

Положение подписей для областей.

property areaPosition: LabelAreaPosition

Режим подписей для областей.

По умолчанию LabelAreaPosition.Horizontal.

property backgroundColor: QColor

Цвет фона.

property backgroundSize: int

Толщина фона в пунктах.

property backgroundType: LabelBackgroundType

Фон подписи.

По умолчанию отсутствует

property color: QColor

Цвет шрифта меток.

property font: QFont

Шрифт.

property horizontalAlign: LabelHorizontalAlign

Горизонтальное выравнивание подписей.

По умолчанию LabelHorizontalAlign.Flat

property lineKeepDirection: bool

Направление текста строится вдоль направления линии.

По умолчанию False.

property lineLayout: LabelLayout

Положение подписей для линий.

property linePosition: LabelLinePosition

Режим подписей для линий.

По умолчанию LabelLinePosition.FollowPath.

property opacity: int

Прозрачность (0..100).

По умолчанию 100 (Непрозрачно).

property overhang: int

Максимальный свес для линии (в %).

property placementPolicy: LabelOverlap

Принцип наложения меток на слой карты.

По умолчанию LabelOverlap.AllowOverlap

property pointLayout: LabelLayout

Положение подписей для точек.

property rangeEnabled: bool

Показывать в пределах.

Если True, используются свойства `rangeMin` и `rangeMax`. По умолчанию False.

property rangeMax: float

Максимальный предел показа с метрах при включенном свойстве `rangeEnabled`.

property rangeMin: float

Минимальный предел показа с метрах при включенном свойстве `rangeEnabled`.

property shadow: bool

Тень.

По умолчанию False

property spacing: bool

Разрядка.

По умолчанию False

property supressDuplicates: bool

Запретить повтор подписей.

Подписи с одинаковым текстом на этом слое будут отображаться один раз. По умолчанию False.

property text: str

Наименование атрибута таблицы либо выражение для метки, которое может основываться на одном или нескольких атрибутах.

property useClip: bool

Использовать динамические подписи.

По умолчанию False.

property visible: bool

Управляет видимостью меток.

Свойства меток**class axipy.render.LabelOverlap**

Стратегия при наложении подписей Label.placementPolicy.

Атрибуты:

AllowOverlap	Допускать перекрытие меток (по умолчанию).
DisallowOverlap	Не допускать перекрытие меток.
OtherPosition	Пробовать найти для метки новую позицию.

class axipy.render.LabelBackgroundType

Фон подписи Label.backgroundType.

Атрибуты:

Empty	Отсутствует.
Frame	Фломастер.
Outline	Кайма.

class axipy.render.LabelLinePosition

Режим подписей для линий Label.linePosition.

Атрибуты:

FollowPath	Вдоль линии.
Horizontal	Горизонтально.
Parallel	Вдоль сегмента.

class axipy.render.LabelAreaPosition

Режим подписей для областей `Label.areaPosition`.

Атрибуты:

Automatic	Автоматически.
Centroid	У центроида.
Horizontal	Горизонтально.
Vertical	Вертикально.

class axipy.render.LabelAreaInterior

Выбор участка для областей `Label.areaInterior`.

Атрибуты:

Centroid	Ближайший к центру.
Max	Наибольший.

class axipy.render.LabelHorizontalAlign

Горизонтальное выравнивание подписи `Label.horizontalAlign`.

Атрибуты:

Begin	Подпись располагается в начале линии.
Center	Середина подписи совпадает с центром линии.
Centroid	Подпись располагается относительно центроида.
End	Подпись располагается в конце линии.
Flat	По центру пологого участка (можно задать угол на вкладке Дополнительно)

LabelLayout - Положение подписей**class axipy.LabelLayout**

Положение подписей `Label`.

Свойства:

<code>offset</code>	Смещение.
<code>position</code>	Расположение подписи.
<code>visible</code>	Видимость подписи для данного типа геометрии.

property offset: QSize

Смещение.

Интервал значений (0..100)

property position: LabelLayoutPosition

Расположение подписи.

property visible: bool

Видимость подписи для данного типа геометрии.

По умолчанию True

class axipy.LabelLayoutPosition

Относительное положение подписи `LabelLayout.position`.

Атрибуты:

Bottom	Снизу.
BottomLeft	Снизу слева.
BottomRight	Снизу справа.
Center	По центру.
Left	Слева.
Right	Справа.
Top	Сверху.
TopLeft	Сверху слева.
TopRight	Сверху справа.

CustomLabelProperties - Свойства выносной метки

class axipy.CustomLabelProperties

Свойства выносной метки. Используется при задании параметров положения метки карты `CustomLabels.set()` или получения их `CustomLabels.get()`.

См.также:

`Map.custom_labels`

Свойства:

<code>angle</code>	Угол поворота текста в градусах.
<code>arrow_style</code>	Стиль указки.
<code>endType</code>	Тип выноски.
<code>expression</code>	Текст.
<code>position</code>	Переопределенное положение метки.
<code>position_align</code>	Относительное положение подписи по отношению к ее позиции <code>position</code> .
<code>text_style</code>	Стиль текстовых объектов.

Специальные методы:

<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

property angle: float

Угол поворота текста в градусах.

property arrow_style: `Optional[Style]`

Стиль указки.

property endType: `CustomLabelEndType`

Тип выноски.

По умолчанию `CustomLabelEndType.EndNone`

property expression: `Optional[str]`

Текст.

Если не задано, используется базовая формула для слоя.

property position: `Optional[Point]`

Переопределенное положение метки.

Если используется по умолчанию, возвращается `None`. Если при задании не указана система координат, используется СК слоя, которому принадлежит метка.

property position_align: `LabelLayoutPosition`

Относительное положение подписи по отношению к ее позиции `position`.

property text_style: `Optional[TextStyle]`

Стиль текстовых объектов.

class `axipy.CustomLabelEndType`

Тип выноски для метки `CustomLabelProperties.endType`.

Атрибуты:

Arrow	Стрелка.
EndNone	Не отображать (по умолчанию)
Line	Линия.

23.4 Legend - Легенда слоя

class `axipy.Legend`

Легенда слоя. Позволяет получить информацию об условных обозначениях на слое. Созданная легенда в дальнейшем может быть помещена на лист отчета `axipy.Report` как `axipy.LegendReportItem` или же расположена в отдельном окне легенд слоев для карты `axipy.Map`.

Параметры

lay - Слой, для которого создается легенда.

Список 16: Пример создания легенды.

```

rangel = RangeThematicLayer("Население")
world.thematic.add(rangel)
# Легенда для тематического слоя
legend = Legend(rangel)
legend.columns = 2
# Зададим стиль заднего фона и окантовки
legend.border_style = LineStyle(3, Qt.red)

```

(continues on next page)

(продолжение с предыдущей страницы)

```

legend.fill_style = PolygonStyle(49, Qt.yellow)
# Изменим описание для первого элемента
item = legend.items[0]
item.title = 'Описание'
legend.items[0] = item
# Заголовок легенды
legend.caption = 'Легенда для слоя'
# Стил заголовка
legend.style_caption = Style.from_mapinfo("Font (\"Arial\", 0, 9, 255)")
# Просмотр всех стилей легенды
for it in legend.items:
    print(it.title, it.visible, it.style.to_mapinfo())
# Изменение позиции элемента легенды
legend.items.move(0, 1)
# Отрисовываем легенду в контексте вместе с картой
image = QImage(800, 600, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.white)
painter = QPainter(image)
context = Context(painter)
legend.position = (100, 50)
legend.draw(context)
...

>>> Описание True Pen (1, 2, 8421504) Brush (2, 16776960)
>>> 55419-166640 True Pen (1, 2, 8421504) Brush (2, 12582656)
>>> 166640-631500 True Pen (1, 2, 8421504) Brush (2, 8453888)
...

```

Свойства:

<code>border_style</code>	Стил используемой окантовки.
<code>caption</code>	Заголовок легенды.
<code>columns</code>	Количество колонок в легенде.
<code>fill_style</code>	Стил заливки заднего фона.
<code>items</code>	Перечень стилей легенды.
<code>position</code>	Положение легенды в контексте рисования.
<code>style_caption</code>	Стил заголовка легенды.
<code>style_subcaption</code>	Стил подзаголовка легенды.
<code>style_text</code>	Стил текстовых подписей.
<code>subcaption</code>	Подзаголовок легенды.

Методы:

<code>draw(context)</code>	Рисует легенду в контексте.
<code>refresh()</code>	Обновляет стили из источника.
<code>to_image(width, height)</code>	Возвращает легенду в виде растра.

property `border_style`: `Style`

Стил используемой окантовки.

Отображается если `has_border` установлено в `True`.

property caption: `str`

Заголовок легенды.

Стиль заголовка задается свойством `style_caption`

property columns: `int`

Количество колонок в легенде.

По умолчанию 1.

draw(context: `Context`)

Рисует легенду в контексте.

Легенду также можно отрисовать совместно с картой в одном контексте (см. `Map.draw()`).

Параметры

context – Контекст рисования.

property fill_style: `Style`

Стиль заливки заднего фона.

property items: `ListLegendItems`

Перечень стилей легенды.

Реализован в виде списка. Для изменения какого-либо параметра необходимо сначала получить элемент, затем поменять требуемое свойство, а затем измененный элемент переназначить.

property position: `Pnt`

Положение легенды в контексте рисования.

refresh()

Обновляет стили из источника.

property style_caption: `Style`

Стиль заголовка легенды.

property style_subcaption: `Style`

Стиль подзаголовка легенды.

property style_text: `Style`

Стиль текстовых подписей.

property subcaption: `str`

Подзаголовок легенды.

Стиль заголовка задается свойством `style_subcaption`

to_image(width: `int`, height: `int`) → `QImage`

Возвращает легенду в виде растра.

Параметры

- **width** – Ширина выходного растра.
- **height** – Высота выходного растра.

23.5 LegendItem - Элемент легенды

class axipy.LegendItem

Элемент легенды.

Свойства:

<code>style</code>	Стиль оформления элемента легенды.
<code>title</code>	Описание элемента легенды.
<code>visible</code>	Видимость элемента легенды.

property style: `Style`

Стиль оформления элемента легенды.

property title: `str`

Описание элемента легенды.

property visible: `bool`

Видимость элемента легенды.

23.6 ListLegendItems - Список элементов легенды

class axipy.ListLegendItems

Элементы легенды.

Специальные методы:

<code>__getitem__(item)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__setitem__(idx, v)</code>	Установка элемента по индексу (<code>obj[i] = value</code>).

`__getitem__(item: int) → Optional[LegendItem]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[LegendItem]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

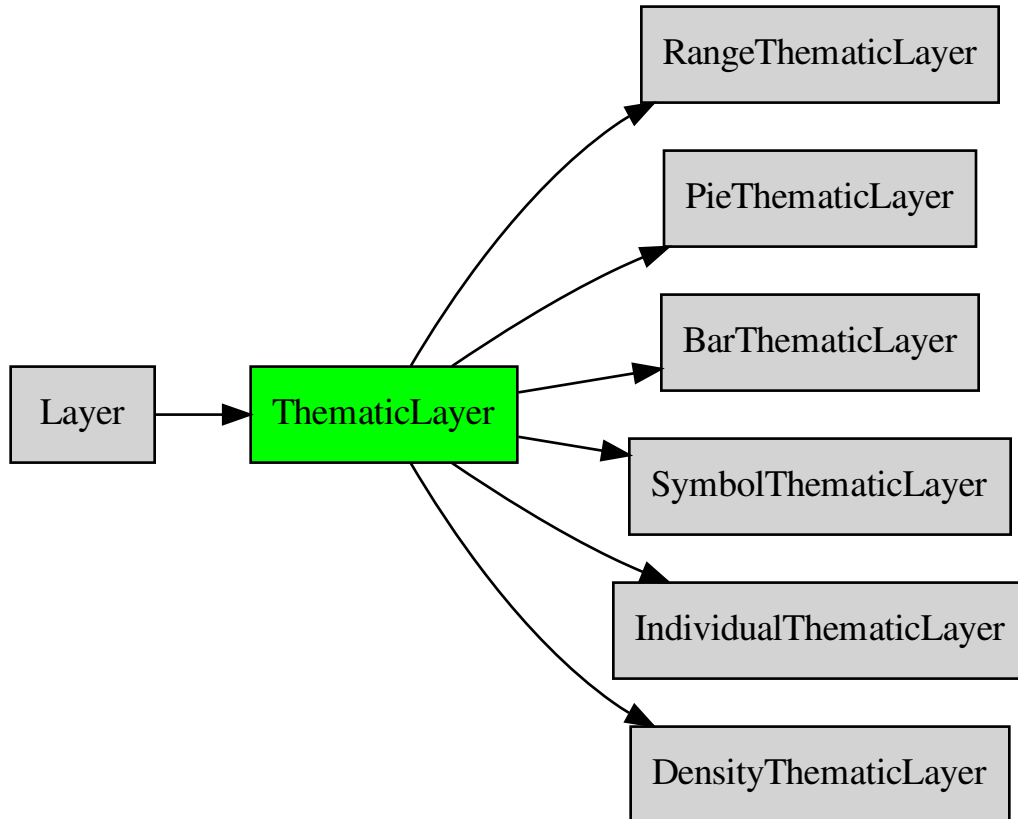
`__setitem__(idx: int, v: LegendItem)`

Установка элемента по индексу (`obj[i] = value`).

23.7 ThematicLayer - Тематика

23.7.1 ThematicLayer - Тематика

Ієрархія класов тематик:



class ахіру.**ThematicLayer**

Базовые класи: [Layer](#)

Абстрактний клас слоя с тематическим оформлением векторного слоя карты на базе атрибутивной информации.

23.7.2 ReallocateThematicColor - Распределение цветов

class axipy.ReallocateThematicColor

Базовые классы: `object`

Поддержка различного рода алгоритмов распределения оформления.

Методы:

<code>assign_gray([minV, maxV])</code>	Распределение в виде градации серого.
<code>assign_monotone(color[, minv, maxv])</code>	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
<code>assign_rainbow([sequential, saturation, value])</code>	Распределение цветов по спектру.
<code>assign_three_colors(colorMin, colorMax, ...)</code>	Цвет, распределенный между тремя заданными цветами (с разрывом).
<code>assign_two_colors(colorMin, colorMax[, useHSV])</code>	Равномерно распределяет оформление по заданным крайним цветам.

assign_gray(minV: `int` = 20, maxV: `int` = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** - Минимальное значение.
- **maxV** - Максимальное значение.

assign_monotone(color: `QColor`, minv: `int` = 20, maxv: `int` = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** - Базовый цвет.
- **minv** - Минимальное значение.
- **maxv** - Максимальное значение.

assign_rainbow(sequential: `bool` = True, saturation: `float` = 90, value: `float` = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** - Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** - Яркость. Задается в интервале (0..100)
- **value** - Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: `QColor`, colorMax: `QColor`, colorBreak: `QColor`, br: `int`, useHSV: `bool` = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** - Цвет нижнего диапазона.

- **colorMax** - Цвет верхнего диапазона.
- **colorBreak** - Цвет на уровне разрыва.
- **br** - Индекс интервала, на котором используется цвет разрыва.
- **useHSV** - Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: Union[QColor, GlobalColor], colorMax: Union[QColor, GlobalColor], useHSV: bool = False)

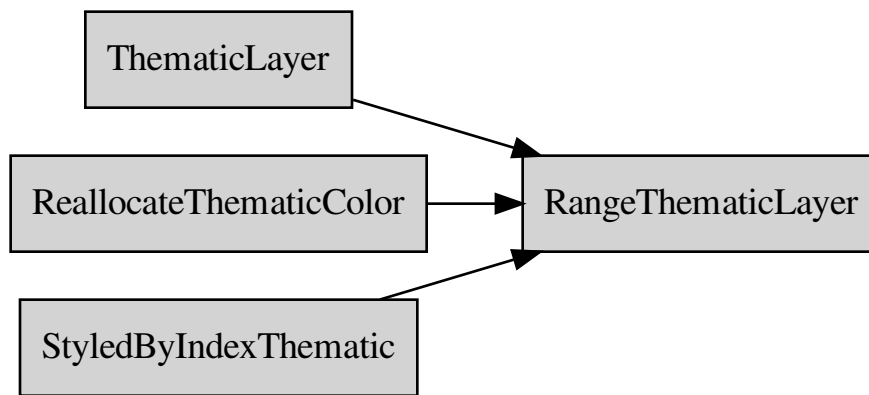
Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** - Цвет нижнего диапазона.
- **colorMax** - Цвет верхнего диапазона.
- **useHSV** - Если True, то будет использоваться схема HSV. В противном случае - RGB.

23.7.3 RangeThematicLayer - Интервалы

Иерархия классов:



class axipy.RangeThematicLayer

Базовые классы: [ThematicLayer](#), [StyledByIndexThematic](#), [ReallocateThematicColor](#)

Тематическое оформление слоя с распределением значений по интервалам. Для распределения цветов по заданным интервалам могут быть использованы функции `assign_*` класса [ReallocateThematicColor](#) в зависимости от требуемых целей.

Параметры

expression - Наименование атрибута таблицы или выражение.

Список 17: Пример создания тематики по интервалам.

```
# Пример создания тематики с последующим добавлением ее к базовому слою `world`
rangel = RangeThematicLayer("Население")
rangel.ranges = 6
rangel.splitType = RangeThematicLayer.EQUAL_COUNT
rangel.assign_two_colors(Qt.red, Qt.cyan)
rangel.style_geometry_type = StyleGeometryType.Linear
world.thematic.add(rangel)
# Пример запроса с последующей заменой
v = world.thematic[0].get_interval_value(2) # Запрос
v = (999, v[1]) # Заменяем минимальное значение для интервала с индексом 2
world.thematic[0].set_interval_value(2, v) # Замена
# Различные виды распределения интервалов тематик по цветам
rangel.assign_two_colors(Qt.red, Qt.yellow)
rangel.assign_three_colors(Qt.yellow, Qt.cyan, Qt.green, 4)
rangel.assign_rainbow()
rangel.assign_gray(80, 100)
```

Классовые методы:

<code>create(dataObject)</code>	Создает слой на базе открытой таблицы или растра.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>ranges</code>	Количество интервалов.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>splitType</code>	Тип распределения значений по интервалам.
<code>style_geometry_type</code>	Тип геометрического объекта для построения легенды.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Атрибуты:

<code>EQUAL_COUNT</code>	Распределение исходя их равного количества объектов в каждом интервале.
<code>EQUAL_INTERVAL</code>	Распределение исходя из равномерности интервалов (по умолчанию).
<code>MANUAL</code>	Ручное распределение значений путем задания пределов вручную.

Методы:

<code>assign_gray([minV, maxV])</code>	Распределение в виде градации серого.
<code>assign_monotone(color[, minv, maxv])</code>	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
<code>assign_rainbow([sequential, saturation, value])</code>	Распределение цветов по спектру.
<code>assign_three_colors(colorMin, colorMax, ...)</code>	Цвет, распределенный между тремя заданными цветами (с разрывом).
<code>assign_two_colors(colorMin, colorMax[, useHSV])</code>	Равномерно распределяет оформление по заданным крайним цветам.
<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.
<code>get_interval_value(idx)</code>	Возвращает предельные значения для указанного интервала в виде пары значений.
<code>get_style(idx)</code>	Стиль для указанного выражения.
<code>set_interval_value(idx, v)</code>	Заменяет предельные значения интервала.
<code>set_style(idx, style)</code>	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`EQUAL_COUNT: Final[int]`

Распределение исходя их равного количества объектов в каждом интервале.

EQUAL_INTERVAL: `Final[int]`

Распределение исходя из равномерности интервалов (по умолчанию).

MANUAL: `Final[int]`

Ручное распределение значений путем задания пределов вручную.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

assign_gray(minV: `int` = 20, maxV: `int` = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_monotone(color: `QColor`, minv: `int` = 20, maxv: `int` = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** – Базовый цвет.
- **minv** – Минимальное значение.
- **maxv** – Максимальное значение.

assign_rainbow(sequential: `bool` = True, saturation: `float` = 90, value: `float` = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** – Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** – Яркость. Задается в интервале (0..100)
- **value** – Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: `QColor`, colorMax: `QColor`, colorBreak: `QColor`, br: `int`, useHSV: `bool` = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **colorBreak** – Цвет на уровне разрыва.
- **br** – Индекс интервала, на котором используется цвет разрыва.

- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: Union[QColor, GlobalColor], colorMax: Union[QColor, GlobalColor], useHSV: bool = False)

Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан **VectorLayer** или **RasterLayer**.

Список 18: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Возвращает источник данных для слоя.

get_bounds() → Rect

Возвращает область, которая содержит все данные слоя.

get_interval_value(idx: int) → Tuple[float, float]

Возвращает предельные значения для указанного интервала в виде пары значений.

Параметры

idx – Индекс диапазона.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры

idx – Порядковый номер выражения.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property ranges: int

Количество интервалов.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

set_interval_value(idx: int, v: Tuple[float, float])

Заменяет предельные значения интервала.

Параметры

- **idx** – Индекс диапазона.
- **v** – Значение в виде пары.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 19: 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)
```

property splitType: int

Тип распределения значений по интервалам.

Таблица 3: Допустимые значения:

Константа	Значение	Описание
EQUAL_INTERVAL		Распределение исходя из равномерности интервалов (по умолчанию).
EQUAL_COUNT		Распределение исходя из равного количества объектов в каждом интервале.
MANUAL	3	Ручное распределение значений путем задания пределов вручную.

property style_geometry_type: StyleGeometryType

Тип геометрического объекта для построения легенды.

Т.е. какой объект для отображения стиля будет представлен в легенде.

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

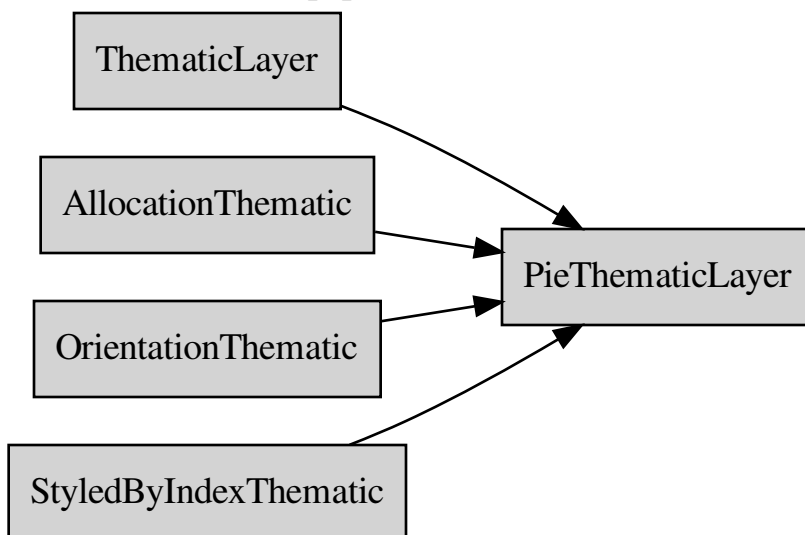
property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.4 PieThematicLayer - Круговые диаграммы

Иерархия классов:



class axipy.PieThematicLayer

Базовые классы: ThematicLayer, AllocationThematic, OrientationThematic, StyledByIndexThematic

Тематика в виде круговых диаграмм.

Параметры

expressions - Наименования атрибутов или выражений в виде списка list.

Список 20: Создание тематики с последующим добавлением ее к базовому слою.

```

pie = PieThematicLayer(["Население", "Мужское", "Женское"])
pie.allocationType = PieThematicLayer.SQRT
style_lay_pie = Style.from_mapinfo("Brush (8, 65535, 0)")
# Заменяем стиль
pie.set_style(0, style_lay_pie)
# Добавляем к основному слою
world.thematic.add(pie)
  
```

Классовые методы:

create(dataObject)

Создает слой на базе открытой таблицы или растра.

Свойства:

allocationType	Тип распределения значений.
coordsystem	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
data_object	Возвращает источник данных для слоя.
is_valid	Возвращает признак валидности слоя.
max_zoom	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
min_zoom	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
opacity	Устанавливает или возвращает прозрачность слоя в составе карты.
orientationType	Ориентация относительно центроида.
selectable	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
startAngle	Начальный угол отсчета диаграммы.
title	Устанавливает или возвращает наименование слоя.
visible	Устанавливает или возвращает признак видимости слоя.
zoom_restrict	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Атрибуты:

CENTER	Диаграмма рисуется по центру (по умолчанию)
DOWN	Диаграмма выравнивается по нижнему краю.
LEFT	Диаграмма выравнивается по левому краю.
LEFT_DOWN	Диаграмма выравнивается по нижнему левому краю.
LEFT_UP	Диаграмма выравнивается по левому верхнему краю.
LINEAR	Линейное (по умолчанию)
LOG10	Логарифмическое.
RIGHT	Диаграмма выравнивается по правому краю.
RIGHT_DOWN	Диаграмма выравнивается по нижнему правому краю.
RIGHT_UP	Диаграмма выравнивается по верхнему правому краю.
SQRT	Квадратичное.
UP	Диаграмма выравнивается по верхнему краю.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.
<code>get_style(idx)</code>	Стиль для указанного выражения.
<code>set_style(idx, style)</code>	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

CENTER

Диаграмма рисуется по центру (по умолчанию)

DOWN

Диаграмма выравнивается по нижнему краю.

LEFT

Диаграмма выравнивается по левому краю.

LEFT_DOWN

Диаграмма выравнивается по нижнему левому краю.

LEFT_UP

Диаграмма выравнивается по левому верхнему краю.

LINEAR

Линейное (по умолчанию)

LOG10

Логарифмическое.

RIGHT

Диаграмма выравнивается по правому краю.

RIGHT_DOWN

Диаграмма выравнивается по нижнему правому краю.

RIGHT_UP

Диаграмма выравнивается по верхнему правому краю.

SQRT

Квадратичное.

UP

Диаграмма выравнивается по верхнему краю.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property allocationType: int

Тип распределения значений.

Таблица 4: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан `VectorLayer` или `RasterLayer`.

Список 21: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Возвращает источник данных для слоя.

get_bounds() → Rect

Возвращает область, которая содержит все данные слоя.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры

idx – Порядковый номер выражения.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property orientationType: int

Ориентация относительно центроида.

Таблица 5: 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

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** - Индекс.
- **style** - Назначаемый стиль.

Список 22: 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)
```

property startAngle: int

Начальный угол отсчета диаграммы.

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

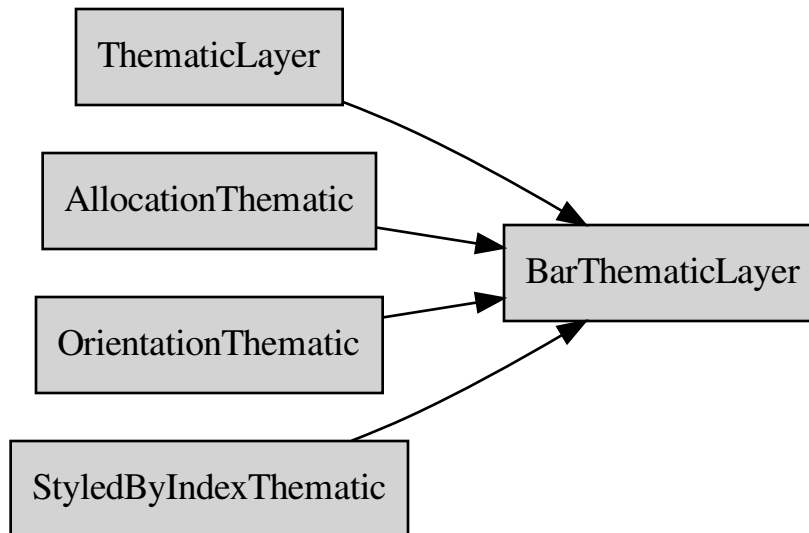
property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.5 BarThematicLayer - Столбчатые диаграммы

Иерархия классов:



class axipy.BarThematicLayer

Базовые классы: `ThematicLayer`, `AllocationThematic`, `OrientationThematic`, `StyledByIndexThematic`

Тематика в виде столбчатых диаграмм.

Параметры

expressions - Наименования атрибутов или выражений в виде списка `list`.

Список 23: Создание тематики с последующим добавлением ее к базовому слою.

```

bar = BarThematicLayer(["Население", "Мужское", "Женское"])
# Добавляем к основному слою
world.thematic.add(bar)
  
```

Классовые методы:

`create(dataObject)`

Создает слой на базе открытой таблицы или раstra.

Свойства:

allocationType	Тип распределения значений.
coordsystem	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
data_object	Возвращает источник данных для слоя.
isStacked	Расположение столбчатой диаграммы в виде стопки, если True.
is_valid	Возвращает признак валидности слоя.
max_zoom	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
min_zoom	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
opacity	Устанавливает или возвращает прозрачность слоя в составе карты.
orientationType	Ориентация относительно центроида.
selectable	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
title	Устанавливает или возвращает наименование слоя.
visible	Устанавливает или возвращает признак видимости слоя.
zoom_restrict	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Атрибуты:

CENTER	Диаграмма рисуется по центру (по умолчанию)
DOWN	Диаграмма выравнивается по нижнему краю.
LEFT	Диаграмма выравнивается по левому краю.
LEFT_DOWN	Диаграмма выравнивается по нижнему левому краю.
LEFT_UP	Диаграмма выравнивается по левому верхнему краю.
LINEAR	Линейное (по умолчанию)
LOG10	Логарифмическое.
RIGHT	Диаграмма выравнивается по правому краю.
RIGHT_DOWN	Диаграмма выравнивается по нижнему правому краю.
RIGHT_UP	Диаграмма выравнивается по верхнему правому краю.
SQRT	Квадратичное.
UP	Диаграмма выравнивается по верхнему краю.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.
<code>get_style(idx)</code>	Стиль для указанного выражения.
<code>set_style(idx, style)</code>	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

CENTER

Диаграмма рисуется по центру (по умолчанию)

DOWN

Диаграмма выравнивается по нижнему краю.

LEFT

Диаграмма выравнивается по левому краю.

LEFT_DOWN

Диаграмма выравнивается по нижнему левому краю.

LEFT_UP

Диаграмма выравнивается по левому верхнему краю.

LINEAR

Линейное (по умолчанию)

LOG10

Логарифмическое.

RIGHT

Диаграмма выравнивается по правому краю.

RIGHT_DOWN

Диаграмма выравнивается по нижнему правому краю.

RIGHT_UP

Диаграмма выравнивается по верхнему правому краю.

SQRT

Квадратичное.

UP

Диаграмма выравнивается по верхнему краю.

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property allocationType: int

Тип распределения значений.

Таблица 6: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан `VectorLayer` или `RasterLayer`.

Список 24: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: DataObject

Возвращает источник данных для слоя.

get_bounds() → Rect

Возвращает область, которая содержит все данные слоя.

get_style(idx: int) → Style

Стиль для указанного выражения.

Параметры

idx – Порядковый номер выражения.

property isStacked: bool

Расположение столбчатой диаграммы в виде стопки, если True.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property orientationType: int

Ориентация относительно центроида.

Таблица 7: 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

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** - Индекс.
- **style** - Назначаемый стиль.

Список 25: 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)
```

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

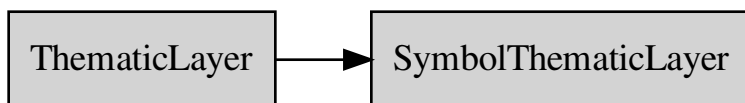
property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.6 SymbolThematicLayer - Знаки

Иерархия классов:



class axipy.SymbolThematicLayer

Базовые классы: [ThematicLayer](#)

Тематический слой с распределением по интервалам и с градуировкой символа по размеру.

Параметры

expression – Наименование атрибута или выражение.

Список 26: 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)
```

Классовые методы:

<code>create(dataObject)</code>	Создает слой на базе открытой таблицы или раstra.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>defaultStyle</code>	Стиль по умолчанию для оформления знаков.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>maxHeight</code>	Максимальная высота символа.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>minHeight</code>	Минимальная высота символа.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: DataObject) → Layer

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан `VectorLayer` или `RasterLayer`.

Список 27: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: Signal

Сигнал об изменении контента слоя.

Тип результата

`Signal[]`

property data_object: DataObject

Возвращает источник данных для слоя.

property defaultStyle: Style

Стиль по умолчанию для оформления знаков.

get_bounds() → [Rect](#)

Возвращает область, которая содержит все данные слоя.

property is_valid: bool

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property maxHeight: int

Максимальная высота символа.

property max_zoom: float

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property minHeight: int

Минимальная высота символа.

property min_zoom: float

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

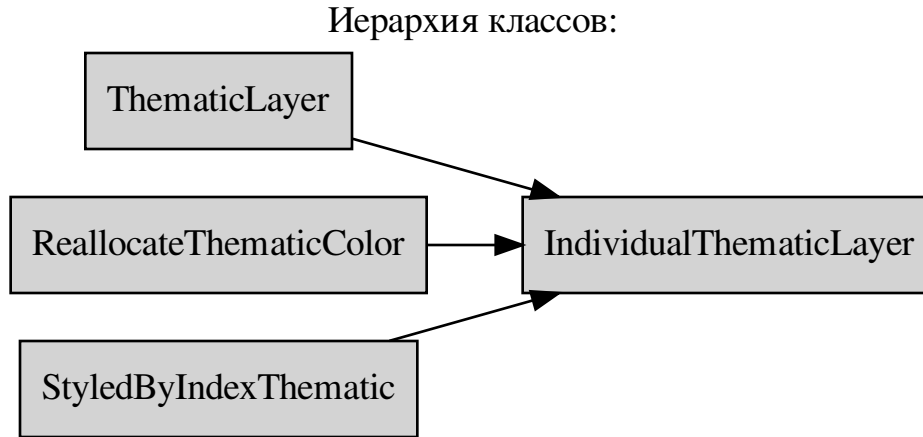
```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.7 IndividualThematicLayer - Индивидуальные значения



class axipy.**IndividualThematicLayer**

Базовые классы: `ThematicLayer`, `StyledByIndexThematic`, `ReallocateThematicColor`

Тематический слой с распределением стилей по индивидуальным значениям.

Параметры

expression – Наименование атрибута или выражение.

Список 28: Создание тематики с последующим добавлением ее к базовому слою.

```
individual = IndividualThematicLayer("Страна")
individual.assign_rainbow()
world.thematic.add(individual)
# Поменяем стиль оформления
individual.set_style(0, PolygonStyle(45, Qt.blue))
```

Классовые методы:

<code>create(dataObject)</code>	Создает слой на базе открытой таблицы или раstra.

Свойства:

<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>count</code>	Количество значений в тематике.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>style_geometry_type</code>	Тип геометрического объекта для построения легенды.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>assign_gray([minV, maxV])</code>	Распределение в виде градации серого.
<code>assign_monotone(color[, minv, maxv])</code>	Монотонная заливка разной яркости (оттенки красного, синего и т.п.).
<code>assign_rainbow([sequential, saturation, value])</code>	Распределение цветов по спектру.
<code>assign_three_colors(colorMin, colorMax, ...)</code>	Цвет, распределенный между тремя заданными цветами (с разрывом).
<code>assign_two_colors(colorMin, colorMax[, useHSV])</code>	Равномерно распределяет оформление по заданным крайним цветам.
<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.
<code>get_style(idx)</code>	Стиль для указанного выражения.
<code>get_value(idx)</code>	Выражение по указанному индексу.
<code>set_style(idx, style)</code>	Установка стиля оформления для выражения по его индексу в списке выражений.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__len__() → int`
Длина объекта (функция `len()`).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`assign_gray`(minV: int = 20, maxV: int = 80)

Распределение в виде градации серого. Значение задается в интервале (0..100) от черного до белого.

Параметры

- **minV** – Минимальное значение.
- **maxV** – Максимальное значение.

assign_monotone(color: QColor, minv: int = 20, maxv: int = 80)

Монотонная заливка разной яркости (оттенки красного, синего и т.п.). Цветовая схема HSL. Максимальное и минимальное значения задаются в интервале (0..100).

Параметры

- **color** – Базовый цвет.
- **minv** – Минимальное значение.
- **maxv** – Максимальное значение.

assign_rainbow(sequential: bool = True, saturation: float = 90, value: float = 90)

Распределение цветов по спектру. Цветовая схема HSV.

Параметры

- **sequential** – Если True, то последовательное распределение цветов. В противном случае распределение случайно.
- **saturation** – Яркость. Задается в интервале (0..100)
- **value** – Насыщенность. Задается в интервале (0..100)

assign_three_colors(colorMin: QColor, colorMax: QColor, colorBreak: QColor, br: int, useHSV: bool = True)

Цвет, распределенный между тремя заданными цветами (с разрывом).

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **colorBreak** – Цвет на уровне разрыва.
- **br** – Индекс интервала, на котором используется цвет разрыва.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

assign_two_colors(colorMin: Union[QColor, GlobalColor], colorMax: Union[QColor, GlobalColor], useHSV: bool = False)

Равномерно распределяет оформление по заданным крайним цветам.

Параметры

- **colorMin** – Цвет нижнего диапазона.
- **colorMax** – Цвет верхнего диапазона.
- **useHSV** – Если True, то будет использоваться схема HSV. В противном случае - RGB.

property coordsystem: CoordSystem

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

property count: int

Количество значений в тематике.

classmethod create(dataObject: [DataObject](#)) → [Layer](#)

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан [VectorLayer](#) или [RasterLayer](#).

Список 29: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: [Signal](#)

Сигнал об изменении контента слоя.

Тип результата

[Signal\[\]](#)

property data_object: [DataObject](#)

Возвращает источник данных для слоя.

get_bounds() → [Rect](#)

Возвращает область, которая содержит все данные слоя.

get_style(idx: [int](#)) → [Style](#)

Стиль для указанного выражения.

Параметры

idx – Порядковый номер выражения.

get_value(idx: [int](#)) → [Any](#)

Выражение по указанному индексу.

Параметры

idx – Индекс.

property is_valid: [bool](#)

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: [float](#)

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property min_zoom: [float](#)

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном zoom_restrict=True.

property need_redraw: [Signal](#)

Сигнал о необходимости перерисовать слой.

Тип результата

[Signal\[\]](#)

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

set_style(idx: int, style: Style)

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 30: 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)
```

property style_geometry_type: StyleGeometryType

Тип геометрического объекта для построения легенды.

Т.е. какой объект для отображения стиля будет представлен в легенде.

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

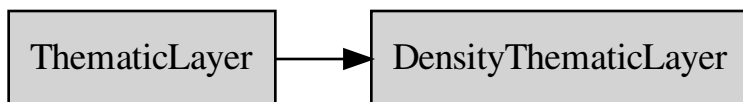
property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.8 DensityThematicLayer - Плотность точек

Иерархия классов:



class axipy.DensityThematicLayer

Базовые классы: [ThematicLayer](#)

Тематический слой с заполнением полигональных объектов точками, плотность которых зависит от вычисленного значения по выражению.

Параметры

expression – Наименование атрибута или выражение.

Список 31: Создание тематики с последующим добавлением ее к базовому слою.

```

density = DensityThematicLayer('Население')
density.pointForMaximum = 500
density.color = Qt.red
density.size = 1
world.thematic.add(density)
  
```

Классовые методы:

[create](#)(dataObject)

Создает слой на базе открытой таблицы или растра.

Свойства:

<code>color</code>	Цвет точек.
<code>coordsystem</code>	Возвращает координатную систему, в которой находятся данные, отображаемые слоем.
<code>data_object</code>	Возвращает источник данных для слоя.
<code>is_valid</code>	Возвращает признак валидности слоя.
<code>max_zoom</code>	Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.
<code>min_zoom</code>	Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.
<code>opacity</code>	Устанавливает или возвращает прозрачность слоя в составе карты.
<code>pointForMaximum</code>	Количество точек для максимального значения.
<code>selectable</code>	Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.
<code>size</code>	Размер точек.
<code>title</code>	Устанавливает или возвращает наименование слоя.
<code>visible</code>	Устанавливает или возвращает признак видимости слоя.
<code>zoom_restrict</code>	Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Методы:

<code>get_bounds()</code>	Возвращает область, которая содержит все данные слоя.

Сигналы:

<code>data_changed</code>	Сигнал об изменении контента слоя.
<code>need_redraw</code>	Сигнал о необходимости перерисовать слой.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

__repr__() → *str*

Официальное строковое представление объекта при вызове `repr()`.

__str__() → *str*

Неформальное строковое представление объекта при вызове `str` или `print()`.

property color: *QColor*

Цвет точек.

property coordsystem: *CoordSystem*

Возвращает координатную систему, в которой находятся данные, отображаемые слоем.

classmethod create(dataObject: *DataObject*) → *Layer*

Создает слой на базе открытой таблицы или растра.

Параметры

dataObject – Таблица или растр. В зависимости от переданного объекта будет создан *VectorLayer* или *RasterLayer*.

Список 32: Пример создания слоя на базе файла.

```
# Векторный слой
table = provider_manager.openfile(filepath)
vector_layer = Layer.create(table)
# Подпишемся на обновление контента слоя
vector_layer.need_redraw.connect(lambda: print('Update layer'))
```

property data_changed: *Signal*

Сигнал об изменении контента слоя.

Тип результата

Signal[]

property data_object: *DataObject*

Возвращает источник данных для слоя.

get_bounds() → *Rect*

Возвращает область, которая содержит все данные слоя.

property is_valid: *bool*

Возвращает признак валидности слоя.

Слой мог быть удален, как пример, в связи с закрытием таблицы.

property max_zoom: *float*

Устанавливает или возвращает максимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном `zoom_restrict=True`.

property min_zoom: *float*

Устанавливает или возвращает минимальную ширину окна, при которой слой отображается на карте.

Учитывается только при установленном `zoom_restrict=True`.

property need_redraw: Signal

Сигнал о необходимости перерисовать слой.

Тип результата

Signal[]

property opacity: int

Устанавливает или возвращает прозрачность слоя в составе карты.

Доступные значения от 0 до 100.

property pointForMaximum: int

Количество точек для максимального значения.

property selectable: bool

Устанавливает или возвращает признак доступности выбора объектов слоя, если это поддерживается.

property size: float

Размер точек.

property title: str

Устанавливает или возвращает наименование слоя.

property visible: bool

Устанавливает или возвращает признак видимости слоя.

Выключение видимости верхнего слоя для активной карты:

```
if axipy.view_manager.active is not None:
    axipy.view_manager.active.map.layers[0].visible = False
```

property zoom_restrict: bool

Устанавливает или возвращает признак, будет ли использоваться ограничение по отображению.

Если установлено True, то для ограничения отображения слоя в зависимости от масштаба используются значения свойств zoom_min и zoom_max.

23.7.9 AllocationThematic - Метод распределения значений для диаграмм

class axipy.AllocationThematic

Метод распределения значений для диаграмм.

Свойства:

<code>allocationType</code>	Тип распределения значений.

Атрибуты:

<code>LINEAR</code>	Линейное (по умолчанию)
<code>LOG10</code>	Логарифмическое.
<code>SQRT</code>	Квадратичное.

LINEAR

Линейное (по умолчанию)

LOG10

Логарифмическое.

SQRT

Квадратичное.

property allocationType: int

Тип распределения значений.

Таблица 8: Допустимые значения.

Константа	Значение	Описание
LINEAR	1	Линейное (по умолчанию)
SQRT	2	Квадратичное
LOG10	3	Логарифмическое

23.7.10 OrientationThematic - Ориентация для диаграмм

class axipy.OrientationThematic

Ориентация тематического представления относительно центроида объекта.

Свойства:

orientationType	Ориентация относительно центроида.

Атрибуты:

CENTER	Диаграмма рисуется по центру (по умолчанию)
DOWN	Диаграмма выравнивается по нижнему краю.
LEFT	Диаграмма выравнивается по левому краю.
LEFT_DOWN	Диаграмма выравнивается по нижнему левому краю.
LEFT_UP	Диаграмма выравнивается по левому верхнему краю.
RIGHT	Диаграмма выравнивается по правому краю.
RIGHT_DOWN	Диаграмма выравнивается по нижнему правому краю.
RIGHT_UP	Диаграмма выравнивается по верхнему правому краю.
UP	Диаграмма выравнивается по верхнему краю.

CENTER

Диаграмма рисуется по центру (по умолчанию)

DOWN

Диаграмма выравнивается по нижнему краю.

LEFT

Диаграмма выравнивается по левому краю.

LEFT_DOWN

Диаграмма выравнивается по нижнему левому краю.

LEFT_UP

Диаграмма выравнивается по левому верхнему краю.

RIGHT

Диаграмма выравнивается по правому краю.

RIGHT_DOWN

Диаграмма выравнивается по нижнему правому краю.

RIGHT_UP

Диаграмма выравнивается по верхнему правому краю.

UP

Диаграмма выравнивается по верхнему краю.

property orientationType: int

Ориентация относительно центра.

Таблица 9: 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

23.7.11 StyledByIndexThematic - Стил заливки

class axipy.StyledByIndexThematic

Поддержка набора индексированных стилей.

Методы:

<code>get_style(idx)</code>	Стиль для указанного выражения.
<code>set_style(idx, style)</code>	Установка стиля оформления для выражения по его индексу в списке выражений.

`get_style(idx: int) → Style`

Стиль для указанного выражения.

Параметры

idx – Порядковый номер выражения.

`set_style(idx: int, style: Style)`

Установка стиля оформления для выражения по его индексу в списке выражений.

Параметры

- **idx** – Индекс.
- **style** – Назначаемый стиль.

Список 33: 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)
```

23.8 Report - Отчет

23.8.1 Report - Отчет

`class axipy.Report`

План отчета для последующей печати.

Список 34: Пример создания пустого отчета и вывод его в pdf.

```
from PySide2.QtPrintSupport import QPrinter
printer = QPrinter()
printer.setPageSize(QPageSize(QPageSize.A4))
printer.setOutputFormat(QPrinter.PdfFormat)
printer.setOutputFileName(filepath)
painterReport = QPainter(printer)
contextReport = Context(painterReport)
report = Report(printer)
report.horizonta_pages = 2
# Здесь добавляются элементы отчета
report.draw(contextReport)
```

См.также:

[Отчет](#)

Свойства:

<code>horizontal_pages</code>	Количество страниц отчета по горизонтали.
<code>items</code>	Элементы отчета.
<code>name</code>	Наименование отчета.
<code>page_size</code>	Размеры одного листа отчета.
<code>unit</code>	Единицы измерения в отчете.
<code>vertical_pages</code>	Количество страниц отчета по вертикали.

Методы:

<code>draw(context)</code>	Выводит отчета в заданном контексте.
<code>fill_on_pages()</code>	Максимально заполняет страницу(ы) отчета.
<code>fit_pages()</code>	Подгоняет число страниц отчета под размер существующих элементов отчета.

Сигналы:

<code>need_redraw</code>	Сигнал о необходимости перерисовки части или всего отчета.

draw(context: `Context`)

Выводит отчета в заданном контексте.

Параметры

context – Контекст, в котором будет отрисован отчет.

fill_on_pages()

Максимально заполняет страницу(ы) отчета.

При этом элементы отчета пропорционально масштабируются.

fit_pages()

Подгоняет число страниц отчета под размер существующих элементов отчета.

При этом параметры элементов отчета не меняются.

property horizontal_pages: int

Количество страниц отчета по горизонтали.

property items: ReportItems

Элементы отчета.

property name: str

Наименование отчета.

property need_redraw: Signal

Сигнал о необходимости перерисовки части или всего отчета.

Параметры

rect – Часть отчета, которую необходимо обновить.

Тип результата

Signal[QRectF]

property page_size: QSizeF

Размеры одного листа отчета.

property unit: LinearUnit

Единицы измерения в отчете.

property vertical_pages: int

Количество страниц отчета по вертикали.

23.8.2 ReportItems - Список элементов отчета

class axipy.ReportItems

Список элементов отчета.

Свойства:

<code>count</code>	Количество элементов отчета в текущем отчете на данный момент.

Методы:

<code>add(item)</code>	Добавляет новый элемент в отчет.
<code>at(idx)</code>	Возвращает элемент отчета по его индексу.
<code>remove(idx)</code>	Удаляет элемент по его индексу.

Специальные методы:

<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__getitem__(idx: int) → Optional[ReportItem]`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[ReportItem]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

add(item: ReportItem)

Добавляет новый элемент в отчет.

Параметры

item – Вставляемый элемент

at(idx: int) → Optional[ReportItem]

Возвращает элемент отчета по его индексу.

Параметры

idx – Индекс.

Результат

Элемент отчета. Возвращает None в случае, если не найдено.

property count: int

Количество элементов отчета в текущем отчете на данный момент.

remove(idx: int)

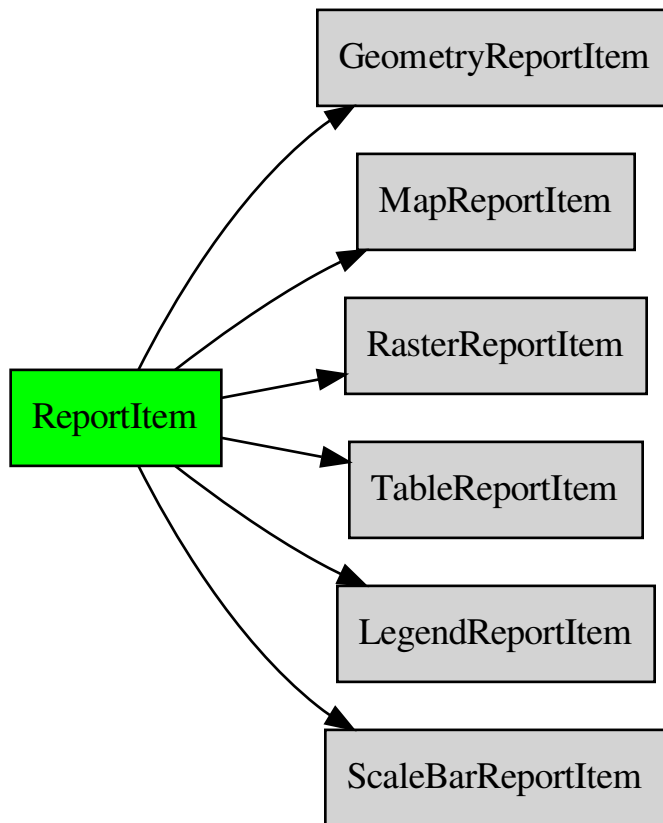
Удаляет элемент по его индексу. Если индекс корректен, элемент будет удален.

Параметры

idx – Индекс удаляемого элемента.

23.8.3 ReportItem - Элемент отчета

Элементы отчета:



class axiру.ReportItem

Базовый класс элемента отчета.

Свойства:

<code>border_style</code>	Стиль обводки элемента отчета.
<code>fill_style</code>	Стиль заливки элемента отчета.
<code>rect</code>	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`
Оператор равенства (==).

property border_style: Style
Стиль обводки элемента отчета.

property fill_style: Style
Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF]) → bool
Пересекается ли с переданным прямоугольником.

Параметры
checkRect – Прямоугольник для анализа.

property rect: Rect
Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

update()
Перерисовка элемента отчета.

23.8.4 GeometryReportItem - Элемент отчета: геометрия

class axipy.GeometryReportItem

Базовые классы: ReportItem

Элемент отчета типа геометрия.

Список 35: Пример создания полигона и добавления его в отчет.

```
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10, 10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
report.items.add(geomItem)
```

Список 36: Пример создания текста и добавления его в отчет.

```
r = Rect(8, 6, 14, 7)
txt = Text("Пример текста", r)
txt.angle = 20
style = Style.from_mapinfo('Font ("Times New Roman", 512, 0, 16711680, 16776960)')
geomItem = GeometryReportItem()
geomItem.style = style
geomItem.geometry = txt
report.items.add(geomItem)
```

Свойства:

<code>border_style</code>	Стиль обводки элемента отчета.
<code>fill_style</code>	Стиль заливки элемента отчета.
<code>geometry</code>	Геометрическое представление объекта.
<code>rect</code>	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
<code>style</code>	Стиль геометрического представления объекта.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`
Оператор равенства (==).

property border_style: Style
Стиль обводки элемента отчета.

property fill_style: Style
Стиль заливки элемента отчета.

property geometry: Geometry
Геометрическое представление объекта.

intersects(checkRect: Union[Rect, QRectF]) → bool
Пересекается ли с переданным прямоугольником.

Параметры

checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

property style: Style

Стиль геометрического представления объекта.

update()

Перерисовка элемента отчета.

23.8.5 MapReportItem - Элемент отчета: карта

class axipy.MapReportItem

Базовые классы: [ReportItem](#)

Элемент отчета, основанный на созданной ранее карте.

Примечание: Перед созданием элемента отчета необходимо предварительно создать карту, на основе которой будет создан элемент отчета.

Список 37: Пример создания карты и добавления ее в отчет.

```
map_ = Map([world])
mapItem = MapReportItem(Rect(10, 110, 200, 210), map_)
mapItem.center = (100, 100)
mapItem.scale = 200000000
report.items.add(mapItem)
```

Конструктор класса:

<code>__init__(rect, map[, coordsystem])</code>	Конструктор класса.

Свойства:

<code>border_style</code>	Стиль обводки элемента отчета.
<code>center</code>	Центр карты в координатах карты.
<code>clip</code>	Геометрия обрезки карты.
<code>coordsystem</code>	Система координат карты.
<code>fill_style</code>	Стиль заливки элемента отчета.
<code>map_rect</code>	Прямоугольник карты в единицах измерения карты.
<code>rect</code>	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
<code>scale</code>	Текущее значение масштаба карты.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>map()</code>	Возвращает элемент типа карта, на основании которой создается элемент отчета.
<code>show_all()</code>	Меняет масштаб карты чтобы показать ее полностью.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`

Оператор равенства (==).

`__init__(rect: Union[Rect, QRectF], map: Map, coordsystem: Optional[CoordSystem] = None)`

Конструктор класса.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **map** – Карта, на базе которой будет создан элемент отчета.
- **coordsystem** – Система координат

property border_style: Style

Стиль обводки элемента отчета.

property center: Pnt

Центр карты в координатах карты.

property clip: ClipGeometry

Геометрия обрезки карты. Устанавливается геометрия, в рамках которой будет отрисована карта. За пределами отрисовка производиться не будет. Обрабатываются только площадные объекты. Так-же допустимо устанавливать коллекции.

Пример установки обрезки по заданной геометрии:

```
reportview = axipy.view_manager.active
if isinstance(reportview, axipy.ReportView):
    rectangle = axipy.Rectangle(axipy.Rect(-50, -50, 50, 50))
    if isinstance(reportview.report.items[0], axipy.MapReportItem):
        reportview.report.items[0].clip.geometry = rectangle
```

Если установить None, то режим сбрасывается:

```
axipy.view_manager.active.report.items[0].clip.geometry = None
```

Если необходимо сохранить объект, то можно просто отключить режим:

```
axipy.view_manager.active.report.items[0].clip.status = False
```

property coordsystem: [CoordSystem](#)

Система координат карти.

property fill_style: [Style](#)

Стиль заливки елемента звіту.

intersects(checkRect: [Union\[Rect, QRectF\]](#)) → [bool](#)

Пересікається чи з переданим прямокутником.

Параметри

checkRect – Прямокутник для аналізу.

map() → [Map](#)

Возвращает элемент типа карты, на основании которой создается элемент отчета.

property map_rect: [Rect](#)

Прямокутник карти в одиницях вимірювання карти.

property rect: [Rect](#)

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

property scale: [float](#)

Текущее значение масштаба карты.

show_all()

Меняет масштаб карты чтобы показать ее полностью.

Пример замены масштаба для всех элементов отчета:

```
for item in reportView.report.items:
    if isinstance(item, MapReportItem):
        item.show_all()
```

update()

Перерисовка элемента отчета.

23.8.6 RasterReportItem - Элемент отчета: растр

class `ахіру.RasterReportItem`

Базовые классы: [ReportItem](#)

Элемент отчета, основанный на растре.

Примечание: В качестве источника может быть как локальный файл, расположенный в файловой системе, так и база растра, размещенного на Web ресурсе.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **data** – Путь к растровому файлу или его URL.

Список 38: Пример элемента на базе URL.

```
report = create_report()
rasterReportItem = RasterReportItem(
    Rect(10, 10, 140, 70),
    'https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Gall%E2%80
    ↪%93Peters_projection_SW.jpg'
    '/1280px-Gall%E2%80%93Peters_projection_SW.jpg'
)
report.items.add(rasterReportItem)
```

Список 39: Пример элемента на базе локального файла.

```
rasterReportItem = RasterReportItem(Rect(10, 10, 140, 70), filename)
report.items.add(rasterReportItem)
```

Свойства:

<code>border_style</code>	Стиль обводки элемента отчета.
<code>fill_style</code>	Стиль заливки элемента отчета.
<code>preserve_aspect_ratio</code>	Сохранять пропорции при изменении размеров элемента.
<code>rect</code>	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`
Оператор равенства (==).

property border_style: Style
Стиль обводки элемента отчета.

property fill_style: Style
Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF]) → bool
Пересекается ли с переданным прямоугольником.

Параметры

checkRect – Прямоугольник для анализа.

property preserve_aspect_ratio: bool
Сохранять пропорции при изменении размеров элемента.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

update()

Перерисовка элемента отчета.

23.8.7 TableReportItem - Элемент отчета: таблица

class ахіру.TableReportItem

Базовые классы: [ReportItem](#)

Элемент отчета табличного представления данных.

Примечание: Позволяет отображать как таблицу целиком, так и накладывая дополнительные ограничения при отображении.

Параметры

- **rect** – Размер элемента отчета в единицах измерения отчета.
- **table** – Таблица.

Список 40: Пример.

```
table = provider_manager.openfile(filename)
tableReportItem = TableReportItem(Rect(210, 150, 480, 100), table)
tableReportItem.columns = table.schema.attribute_names[:3] # Берем для показа
↳первые три атрибута
tableReportItem.row_from = 5 # С 5-й строки
tableReportItem.row_count = 4 # Показываем 4 строки
tableReportItem.start_number = 5 # Нумерация с 5
tableReportItem.border_style = LineStyle(3, Qt.red) # Стиль рамки
tableReportItem.fill_style = PolygonStyle(8, 65535) # Стиль фона
report.items.add(tableReportItem)
```

Свойства:

border_style	Стиль обводки элемента отчета.
columns	Перечень наименований для отображения.
fill_style	Стиль заливки элемента отчета.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.
row_count	Количество записей.
row_from	Номер первой строки из таблицы или запроса.
show_row_number	Показывать ли номера строк.
start_number	Нумерация записей.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>refreshValues()</code>	Обновление данных из таблицы.
<code>table()</code>	Базовая таблица или запрос.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`

Оператор равенства (==).

property border_style: Style

Стиль обводки элемента отчета.

property columns: List[str]

Перечень наименований для отображения.

Если задать пустой список, будут отображены все поля таблицы.

property fill_style: Style

Стиль заливки элемента отчета.

intersects(checkRect: Union[Rect, QRectF]) → bool

Пересекается ли с переданным прямоугольником.

Параметры

checkRect – Прямоугольник для анализа.

property rect: Rect

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

refreshValues()

Обновление данных из таблицы.

property row_count: int

Количество записей.

Если указано -1, то берутся все оставшиеся записи.

property row_from: int

Номер первой строки из таблицы или запроса.

property show_row_number: bool

Показывать ли номера строк.

property start_number: int

Нумерация записей.

Порядковый номер первой записи.

table() → Table

Базовая таблица или запрос.

update()

Перерисовка элемента отчета.

23.8.8 LegendReportItem - Элемент отчета: легенда

class axipy.LegendReportItem

Базовые классы: [ReportItem](#)

Элемент отчета, основанный на легенде векторного или тематического слоя.

Параметры

- **rect** - Размер элемента отчета в единицах измерения отчета.
- **legend** - Предварительно созданная легенда. Она может относиться как к векторному, так и к тематическому слою.

Список 41: Пример создания легенды для тематического слоя.

```
range_ = RangeThematicLayer("Население")
world.thematic.add(range_)
legend = Legend(range_)
legend.columns = 2 # Разобьем на 2 колонки
legendReportItem = LegendReportItem(Rect(100, 230, 50, 70), legend) # Элемент
↪ отчета
report.items.add(legendReportItem) # Добавляем в отчет
```

Свойства:

<code>border_style</code>	Стиль обводки элемента отчета.
<code>fill_style</code>	Стиль заливки элемента отчета.
<code>legend</code>	Легенда на базе которой создан элемент отчета.
<code>rect</code>	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

<code>intersects(checkRect)</code>	Пересекается ли с переданным прямоугольником.
<code>update()</code>	Перерисовка элемента отчета.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).

`__eq__(other: object) → bool`
Оператор равенства (==).

property border_style: Style
Стиль обводки элемента отчета.

property fill_style: Style
Стиль заливки элемента отчета.

intersects (checkRect: [Union\[Rect, QRectF\]](#)) → bool

Пересекается ли с переданным прямоугольником.

Параметры

checkRect – Прямоугольник для анализа.

property legend: [Legend](#)

Легенда на базе которой создан элемент отчета.

property rect: [Rect](#)

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

update()

Перерисовка элемента отчета.

23.8.9 ScaleBarReportItem - Элемент отчета: масштабная линейка

class axipy.[ScaleBarReportItem](#)

Базовые классы: [ReportItem](#)

Элемент отчета - масштабная линейка для карты.

Список 42: Пример создания масштабной линейки на базе существующего элемента - карты.

```
scaleBarReportItem = ScaleBarReportItem(Rect(120, 130, 80, 50), mapItem)
report.items.add(scaleBarReportItem)
```

Свойства:

border_style	Стиль обводки элемента отчета.
fill_style	Стиль заливки элемента отчета.
rect	Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

Методы:

intersects (checkRect)	Пересекается ли с переданным прямоугольником.
update ()	Перерисовка элемента отчета.

Специальные методы:

__eq__ (other)	Оператор равенства (==).

[__eq__](#) (other: [object](#)) → bool

Оператор равенства (==).

property border_style: [Style](#)

Стиль обводки элемента отчета.

property fill_style: [Style](#)

Стиль заливки элемента отчета.

intersects(checkRect: [Union\[Rect, QRectF\]](#)) → bool

Пересекается ли с переданным прямоугольником.

Параметры

checkRect – Прямоугольник для анализа.

property rect: [Rect](#)

Размер (ограничивающий прямоугольник) элемента отчета в единицах измерения отчета.

update()

Перерисовка элемента отчета.

23.9 Context - Контекст рисования

class [axipy.Context](#)

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 – 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)
```

Свойства:

clip	Геометрия обрезки карты.
coordsystem	Координатная система.
dpi	Количество точек на дюйм, с которым происходит рисование.
rect	Прямоугольник в координатах карты, который будет отрисован.

property clip: [ClipGeometry](#)

Геометрия обрезки карты. Устанавливается геометрия, в рамках которой будет отрисована карта. За пределами отрисовка производиться не будет. Обработываются только площадные объекты. Также допустимо устанавливать коллекции.

```
# Исходный файл
table = provider_manager.openfile(filepath)
map_ = Map([table])
# Результирующий растр
image = QImage(1000, 1000, QImage.Format_ARGB32_Premultiplied)
image.fill(Qt.transparent)
painter = QPainter(image)
context = Context(painter)
context.coordsystem = table.coordsystem
# Зададим область врезки в координатах СК
v = 1000000
context.clip.geometry = Rectangle(Rect(-v, -v, v, v))
# отрисовываем в растр
map_.draw(context)
# Сохраняем в файл
image.save(out_filepath)
```

property coordsystem: CoordSystem

Координатная система.

Если она не задана, берется наиболее подходящая исходя из текущего контента.

property dpi: float

Количество точек на дюйм, с которым происходит рисование.

Влияет на отрисовку в «реальных» единицах измерения (мм, см, пункты).

property rect: Rect

Прямоугольник в координатах карты, который будет отрисован.

23.10 CustomLabels - Пользовательские метки карты

class ахіру.CustomLabels

Пользовательские метки.

Используется для задания параметров через свойство `Map.custom_labels`.

Методы:

<code>get(layer, id)</code>	Производит запрос параметров.
<code>ids(layer)</code>	Перечень идентификаторов, для которых установлена пользовательская метка.
<code>set(layer, id, properties)</code>	Устанавливает параметры.
<code>set_default_arrow_type(layer, style)</code>	Устанавливает стиль по умолчанию для новых меток слоя.
<code>set_default_end_type(layer, end_type)</code>	Устанавливает тип выноски по умолчанию для новых меток слоя.

get(layer: VectorLayer, id: int) → Optional[CustomLabelProperties]

Производит запрос параметров. Если для данного id не определены, возвращает None.

Параметры

- **layer** – Слой карты.
- **id** – Идентификатор записи.

ids(layer: [VectorLayer](#)) → [List\[int\]](#)

Перечень идентификаторов, для которых установлена пользовательская метка.

set(layer: [VectorLayer](#), id: [int](#), properties: [Optional\[CustomLabelProperties\]](#))

Устанавливает параметры.

Параметры

- **layer** – Слой карты.
- **id** – Идентификатор записи.
- **properties** – Устанавливаемые свойства. Если задать None, существующие параметры будут сброшены.

set_default_arrow_type(layer: [VectorLayer](#), style: [LineStyle](#))

Устанавливает стиль по умолчанию для новых меток слоя.

set_default_end_type(layer: [VectorLayer](#), end_type: [CustomLabelEndType](#))

Устанавливает тип выноски по умолчанию для новых меток слоя.

23.11 ClipGeometry - Обрезка карты

class `ахіру.ClipGeometry`

Обрезка карты по выбранному участку.

Свойства:

buffer	Размер буфера в координатах экрана или же раstra.
geometry	Геометрия обрезки карты.
status	Включение режима обрезки карты geometry .

property `buffer: int`

Размер буфера в координатах экрана или же раstra.

Это может быть необходимо когда объект обрезки имеет стиль оформления по контуру и этот контур необходимо включить.

property `geometry: Optional[Geometry]`

Геометрия обрезки карты. Устанавливается геометрия, в рамках которой будет отрисована карта. За пределами отрисовка производиться не будет. Обрабатываются только площадные объекты. Так-же допустимо устанавливать коллекции.

См.также:

Элемент отчета [MapReportItem.clip](#)

См.также:

Окно карты `MapView.clip`

См.также:

Окно карты `Context.clip`

property status: `bool`

Включение режима обрезки карты `geometry`.

Если устанавливается значение `True` и отсутствует геометрия, то установка не производится.

axipy.gui - Модуль пользовательского интерфейса.

Модуль пользовательского интерфейса.

В данном модуле содержатся классы связанные с пользовательским интерфейсом.

24.1 ViewTool - Инструмент окна карты или отчета

class axipy.ViewTool

Инструмент работы с окном карты или отчета.

Классовые методы:

<code>reset()</code>	Переключает текущий инструмент на инструмент по умолчанию.

Свойства:

<code>cursor</code>	Устанавливает или возвращает курсор инструмента.
<code>is_active</code>	Возвращает признак активности окна, с которым работает инструмент.
<code>view</code>	Возвращает окно карты или отчета, с которым работает инструмент.

Атрибуты:

<code>ACCEPT</code>	При возвращении этого атрибута в обработчике событий, событие будет принято обработчиком и не пойдет дальше.
<code>IGNORE</code>	При возвращении этого атрибута в обработчике событий, событие будет проигнорировано обработчиком и пойдет дальше.

Методы:

<code>can_unload(reason)</code>	Переопределите, чтобы обрабатывать причину выключения инструмента.
<code>get_select_rect(device[, size])</code>	Возвращает прямоугольник в координатах карты для точки на экране.
<code>handleEvent(event)</code>	Переопределите, чтобы использовать первичный обработчик всех событий инструмента.
<code>is_snapped()</code>	Проверяет, сработала ли привязка к элементам карты или отчета для текущего положения указателя мыши.
<code>keyPressEvent(event)</code>	Переопределите, чтобы обрабатывать событие нажатия клавиши клавиатуры.
<code>keyReleaseEvent(event)</code>	Переопределите, чтобы обрабатывать событие отпускания клавиши клавиатуры.
<code>load()</code>	Переопределите, чтобы задать действия при включении инструмента.
<code>mouseDoubleClickEvent(event)</code>	Переопределите, чтобы обрабатывать событие двойного клика мыши.
<code>mouseMoveEvent(event)</code>	Переопределите, чтобы обрабатывать событие перемещения мыши.
<code>mousePressEvent(event)</code>	Переопределите, чтобы обрабатывать событие нажатия клавиши мыши.
<code>mouseReleaseEvent(event)</code>	Переопределите, чтобы обрабатывать событие отпускания клавиши мыши.
<code>on_active_changed()</code>	Переопределите, чтобы обрабатывать смену активности окна, с которым работает инструмент.
<code>paintEvent(event, painter)</code>	Переопределите, чтобы обрабатывать событие отрисовки.
<code>redraw()</code>	Перерисовывает окно карты.
<code>snap()</code>	Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения карты или отчета.
<code>snap_device()</code>	Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения окна карты (виджета).
<code>to_device()</code>	Переводит точки из координат на карте в координаты окна(пиксели).
<code>to_scene()</code>	Переводит точки из координат окна(пикселей) в координаты на карте.
<code>unload()</code>	Переопределите, чтобы задать действия при выключении инструмента.
<code>wheelEvent(event)</code>	Переопределите, чтобы обрабатывать событие колеса мыши.

Классы:

ViewType	Тип окна просмотра

ACCEPT: Literal[True]

При возвращении этого атрибута в обработчике событий, событие будет принято обработчиком и не пойдет дальше.

IGNORE: Literal[False]

При возвращении этого атрибута в обработчике событий, событие будет проигнорировано обработчиком и пойдет дальше.

class ViewType

Тип окна просмотра

Атрибуты:

MAP_VIEW	Окно карты
REPORT_VIEW	Окно отчета

can_unload(reason: DeactivationReason) → bool

Переопределите, чтобы обрабатывать причину выключения инструмента.

Параметры

reason – причина выключения.

Результат

False, чтобы прервать выключение, иначе True.

property cursor: QCursor

Устанавливает или возвращает курсор инструмента.

Первоначально курсор для инструмента можно установить, переопределив метод `load()`:

```
class MyTool(ViewTool):
    def load(self):
        self.cursor = QCursor(Qt.SizeAllCursor)
```

Если же требуется устанавливать различные типы курсора в зависимости от статуса нажатия ПКМ, следует переопределить методы `mousePressEvent()` и `mouseReleaseEvent()` и установить нужное значение там:

```
class MyTool(ViewTool):
    def mousePressEvent(self, event) -> bool:
        if event.button() == Qt.LeftButton:
            self.cursor = QCursor(Qt.SizeAllCursor)
```

get_select_rect(device: QPoint, size: Optional[int] = None) → Rect

Возвращает прямоугольник в координатах карты для точки на экране. Удобно для использования при поиске объектов в месте нажатия мыши.

Параметры

- **device** – Точка в координатах окна.
- **size** – Размер квадрата в пикселях. При `None`, берется из настроек Аксиомы (`axipy.CurrentSettings.SensitiveMouse`)

Результат

Прямоугольник в координатах карты.

Пример:

```
device_point = event.pos()
bbox = self.get_select_rect(device_point, 30)
features = table.items(bbox=bbox)
```

handleEvent(event: *QEvent*) → *bool*

Переопределите, чтобы использовать первичный обработчик всех событий инструмента.

Если событие не блокируется этим обработчиком, то оно будет передано дальше в соответствующий специализированный обработчик *mousePressEvent()*, *keyReleaseEvent()* и прочие в зависимости от типа.

Параметры

event – Событие.

Результат

BlockEvent, чтобы блокировать дальнейшую обработку события.

property is_active: *bool*

Возвращает признак активности окна, с которым работает инструмент.

is_snapped() → *bool*

Проверяет, сработала ли привязка к элементам карты или отчета для текущего положения указателя мыши.

См.также:

snap(), *snap_device()*.

keyPressEvent(event: *QKeyEvent*) → *bool*

Переопределите, чтобы обрабатывать событие нажатия клавиши клавиатуры.

Параметры

event – Событие нажатия клавиши клавиатуры.

Результат

PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. *None* или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

keyReleaseEvent(event: *QKeyEvent*) → *bool*

Переопределите, чтобы обрабатывать событие отпускания клавиши клавиатуры.

Параметры

event – Событие отпускания клавиши клавиатуры.

Результат

PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. *None* или *BlockEvent*, чтобы блокировать дальнейшую обработку события.

load()

Переопределите, чтобы задать действия при включении инструмента.

См.также:

`unload()`.

mouseDoubleClickEvent(event: `QMouseEvent`) → `bool`

Переопределите, чтобы обрабатывать событие двойного клика мыши.

Параметры

event – Событие двойного клика мыши.

Результат

`PassEvent`, чтобы пропустить событие дальше по цепочке обработчиков. `None` или `BlockEvent`, чтобы блокировать дальнейшую обработку события.

mouseMoveEvent(event: `QMouseEvent`) → `bool`

Переопределите, чтобы обрабатывать событие перемещения мыши.

Параметры

event – Событие перемещения мыши.

Результат

`PassEvent` или `None`, чтобы пропустить событие дальше по цепочке обработчиков. `BlockEvent`, чтобы блокировать дальнейшую обработку события.

mousePressEvent(event: `QMouseEvent`) → `bool`

Переопределите, чтобы обрабатывать событие нажатия клавиши мыши.

Параметры

event – Событие нажатия клавиши мыши.

Результат

`PassEvent`, чтобы пропустить событие дальше по цепочке обработчиков. `None` или `BlockEvent`, чтобы блокировать дальнейшую обработку события.

mouseReleaseEvent(event: `QMouseEvent`) → `bool`

Переопределите, чтобы обрабатывать событие отпускания клавиши мыши.

Параметры

event – Событие отпускания клавиши мыши.

Результат

`PassEvent`, чтобы пропустить событие дальше по цепочке обработчиков. `None` или `BlockEvent`, чтобы блокировать дальнейшую обработку события.

on_active_changed()

Переопределите, чтобы обрабатывать смену активности окна, с которым работает инструмент.

См.также:

`is_active`.

paintEvent(event: `QPaintEvent`, painter: `QPainter`)

Переопределите, чтобы обрабатывать событие отрисовки.

Параметры

- **event** – Событие отрисовки.

- **painter** – QPainter для рисования поверх виджета

redraw()

Перерисовывает окно карты.

Создает событие `PySide2.QtGui.QPaintEvent` и помещает его в очередь обработки событий. Аналогично `PySide2.QtWidgets.QWidget.update()`.

static reset()

Переключает текущий инструмент на инструмент по умолчанию. Инструментом по умолчанию является Выбор. В результате будет вызван `unload()` текущего инструмента. Делает то же самое что `axipy.ActionManager.activate_selection_tool()`.

snap() → `Optional[Pnt]`

snap(default_value: `None`) → `Optional[Pnt]`

snap(default_value: `Pnt`) → `Pnt`

Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения карты или отчета.

Параметры

default_value – Значение по умолчанию.

Возвращает значение по умолчанию, если не сработала привязка к элементам карты или отчета.

Пример:

```
point = self.to_scene(event.pos())
current_point = self.snap(point)
```

См.также:

`is_snapped()`, `snap_device()`, `to_scene()`.

snap_device() → `Optional[QPoint]`

snap_device(default_value: `None`) → `Optional[QPoint]`

snap_device(default_value: `QPoint`) → `QPoint`

Возвращает исправленные координаты, если сработала привязка к элементам в единицах измерения окна карты (виджета).

Параметры

default_value – Значение по умолчанию.

Возвращает значение по умолчанию, если не сработала привязка к элементам карты или отчета.

Пример:

```
device_point = event.pos()
current_device_point = self.snap_device(device_point)
```

См.также:

`is_snapped()`, `snap()`.

to_device(scene: `Pnt`) → `QPoint`

to_device(scene: Rect) → QRect

Переводит точки из координат на карте в координаты окна(пиксели).

Параметры

scene – Точки в координатах карты.

Результат

Точки в координатах окна.

to_scene(device: QPoint) → Pnt

to_scene(device: QRect) → Rect

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

Параметры

device – Точки в координатах окна.

Результат

Точки в координатах карты.

unload()

Переопределите, чтобы задать действия при выключении инструмента.

См.также:

`load()`.

property view: Union[MapView, ReportView]

Возвращает окно карты или отчета, с которым работает инструмент.

wheelEvent(event: QWheelEvent) → bool

Переопределите, чтобы обрабатывать событие колеса мыши.

Параметры

event – Событие колеса мыши.

Результат

PassEvent, чтобы пропустить событие дальше по цепочке обработчиков. None или BlockEvent, чтобы блокировать дальнейшую обработку события.

24.2 DeactivationReason - Причина выключения инструмента

class axipy.DeactivationReason

Причина выключения инструмента.

Атрибуты:

ActionClick	Нажатие на действие.
ActionShortcut	Вызов действия комбинацией клавиш.
LayerClick	Нажатие на свойства слоя.
ObjectClose	Закрытие объекта данных.
Unknown	Не определено.
WindowClose	Закрытие окна.

24.3 ActiveToolPanel - Панель активного инструмента

class axipy.ActiveToolPanel

Сервис предоставляющий доступ к панели активного инструмента.

Список 1: Пример использования.

```
service = ActiveToolPanel()
# Любой пользовательский графический элемент
widget = QWidget()

# Создаём обработчик для панели активного инструмента через который
# будем управлять панелью.
tool_panel = service.make_acceptable(
    title="Мой инструмент",
    observer_id=DefaultKeys.SelectionEditable,
    widget=widget)

# Подписываемся на сигнал отправляемый после нажатия на кнопку "Применить" в
# панели
tool_panel.accepted.connect(lambda: print("Применяем изменения"))
```

Чтобы отобразить переданный ранее графический элемент нужно вызвать `axipy.AxipyActiveToolPanelHandlerBase.activate()`. Например при нажатии на пользовательскую кнопку.

Панель активного инструмента созданная через `axipy.ActiveToolPanel.make_acceptable()` по умолчанию содержит кнопку «Применить». По нажатию на эту кнопку отсылается сигнал `axipy.AxipyAcceptableActiveToolHandler.accepted()` который можно обработать в пользовательском инструменте.

Панель активного инструмента созданная через `axipy.ActiveToolPanel.make_custom()` представляет из себя пустой контейнер в который можно поместить пользовательский графический элемент. Это дает больше свободы для реализации управления панелью активного инструмента.

Переданный идентификатор наблюдателя используется для управления видимостью и доступностью панели. Панель активного инструмента сразу закроется как только наблюдатель вернет False.

Методы:

<code>make_acceptable(title, observer_id[, widget])</code>	Создает экземпляр обработчика, через который можно взаимодействовать с панелью активного инструмента.
<code>make_custom(title, observer_id[, widget])</code>	Создает экземпляр обработчика панели активного инструмента.

make_acceptable(title: str, observer_id: Observer, widget: Optional[QWidget] = None) → AxipyAcceptableActiveToolHandler

Создает экземпляр обработчика, через который можно взаимодействовать с панелью активного инструмента. По умолчанию добавляются кнопки «Применить/Отменить».

Параметры

- **title** - Заголовок панели активного инструмента. Обычно это название инструмента.
- **observer_id** - Идентификатор наблюдателя для управления видимостью и доступностью.
- **widget** - Пользовательский виджет который будет отображаться в панели активного инструмента.

make_custom(title: str, observer_id: Observer, widget: Optional[QWidget] = None) → **AxipyCustomActiveToolPanelHandler**

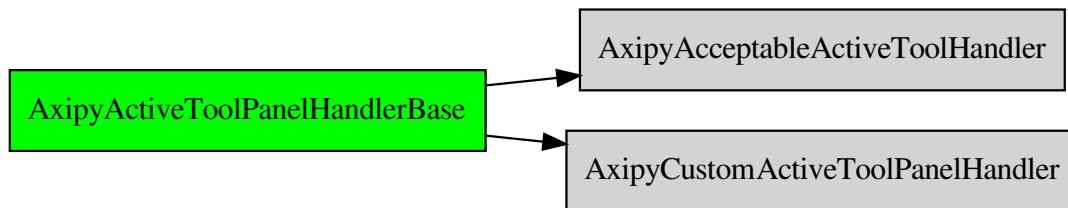
Создает экземпляр обработчика панели активного инструмента. В который можно установить любой пользовательский графический элемент. Используется когда пользователю не нужны предустановленные элементы управления.

Параметры

- **title** - Заголовок панели активного инструмента. Обычно это название инструмента.
- **observer_id** - Идентификатор наблюдателя для управления видимостью и доступностью.
- **widget** - Пользовательский виджет который будет отображаться в панели активного инструмента.

24.4 AxipyActiveToolPanelHandlerBase - Базовый класс обработчика панели активного инструмента

Схема наследования



class axipy.AxipyActiveToolPanelHandlerBase

Базовый класс обработчика панели активного инструмента.

Свойства:

<code>widget</code>	Возвращает пользовательский графический элемент.

Методы:

<code>activate()</code>	Показывает пользовательский графический элемент в панели активного инструмента.
<code>deactivate()</code>	Скрывает пользовательский графический элемент из панели активного инструмента.
<code>set_observer(observer_id)</code>	Метод устанавливает наблюдателя.
<code>set_panel_title(title)</code>	Устанавливает заголовок панели активного инструмента.
<code>set_widget(widget)</code>	Пользовательский графический элемент будет помещен в панель активного инструмента при активации обработчика.

Сигналы:

<code>activated</code>	Сигнал испускается когда обработчик панели активного инструмента становится активным.
<code>deactivated</code>	Сигнал испускается перед тем как обработчик панели активного инструмента перестает быть активным.
<code>panel_was_closed</code>	Сигнал испускается после закрытия панели активного инструмента.

activate()

Показывает пользовательский графический элемент в панели активного инструмента.

property activated: `Signal`

Сигнал испускается когда обработчик панели активного инструмента становится активным.

Тип результата
`Signal[]`

deactivate()

Скрывает пользовательский графический элемент из панели активного инструмента.

property deactivated: `Signal`

Сигнал испускается перед тем как обработчик панели активного инструмента перестает быть активным.

Тип результата
`Signal[]`

property panel_was_closed: `Signal`

Сигнал испускается после закрытия панели активного инструмента.

Тип результата
`Signal[]`

set_observer(observer_id: [Observer](#))

Метод устанавливает наблюдателя. Если наблюдатель сигнализирует, что условия доступности кнопки нарушены, то панель активного инструмента сразу же закрывается.

Параметры

observer_id - Идентификатор наблюдателя для управления видимостью и доступностью

См.также:

Наблюдатели за состоянием инструмента [axipy.ObserverManager](#)

set_panel_title(title: [str](#))

Устанавливает заголовок панели активного инструмента.

Параметры

title - Новый заголовок.

set_widget(widget: [QWidget](#))

Пользовательский графический элемент будет помещен в панель активного инструмента при активации обработчика. Владение графическим элементом передается обработчику. Это значит, что не следует использовать и сохранять где-либо ссылку на этот объект. Для получения графического элемента обратно используйте [axipy.AxipyActiveToolPanelHandlerBase.widget\(\)](#).

property widget: [QWidget](#)

Возвращает пользовательский графический элемент.

Результат

Переданный ранее пользовательский графический элемент.

24.5 AxipyAcceptableActiveToolHandler - Управление панелью активного инструмента с предустановленными кнопками

class [axipy.AxipyAcceptableActiveToolHandler](#)

Базовые классы: [AxipyActiveToolPanelHandlerBase](#)

Обработчик панели активного инструмента, который предоставляет по умолчанию кнопку Применить. При нажатии на эту кнопку испускается сигнал [axipy.AxipyAcceptableActiveToolHandler.accepted\(\)](#).

Методы:

disable()	Отключает доступность блока с кнопкой Применить.
try_enable()	Включает доступность блока с кнопкой 'Применить' если наблюдатель, связанный с панелью активного инструмента, подтверждает доступность.

Сигналы:

<code>accepted</code>	Отсылается после того как пользователь нажал кнопку "Применить" в панели активного инструмента.

property accepted: `Signal`

Отсылается после того как пользователь нажал кнопку «Применить» в панели активного инструмента.

Тип результата

`Signal[]`

`disable()`

Отключает доступность блока с кнопкой Применить. Если инструмент запускает фоновые задачи с использованием `axipy.concurrent.TaskManager`, то следует вызвать эту функцию перед началом выполнения задачи. Иначе у пользователя может быть возможность добавить множество одинаковых задач, несколько раз нажав на кнопку.

`try_enable()`

Включает доступность блока с кнопкой „Применить“ если наблюдатель, связанный с панелью активного инструмента, подтверждает доступность.

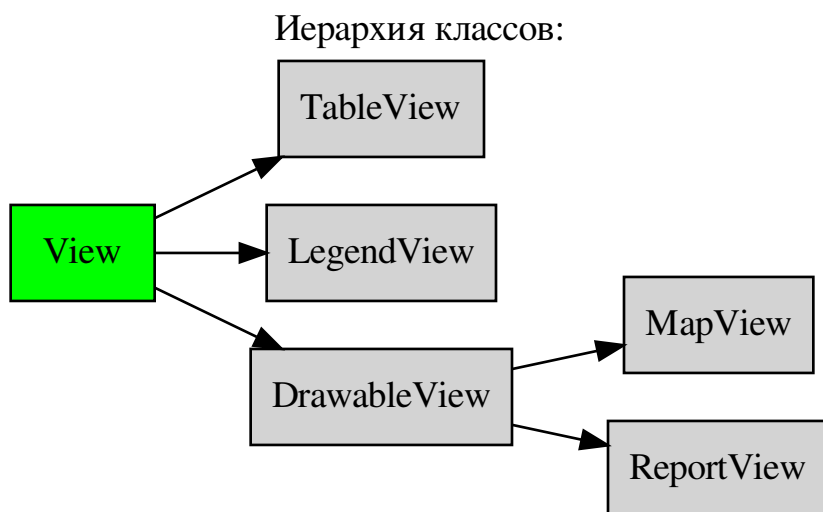
24.6 `AxipyCustomActiveToolPanelHandler` - Управление панелью активного инструмента без предустановленных кнопок управления

`class axipy.AxipyCustomActiveToolPanelHandler`

Базовые классы: `AxipyActiveToolPanelHandlerBase`

Обработчик панели активного инструмента который не предоставляет никаких встроенных по умолчанию элементов управления.

24.7 View - Базовый класс для отображения данных в окне



class axipy.View

Базовый класс для отображения данных в окне.

Свойства:

<code>position</code>	Размер и положение окна.
<code>show_type</code>	Возвращает тип состояния окна.
<code>title</code>	Заголовок окна просмотра.
<code>widget</code>	Виджет, соответствующий содержимому окна.

Методы:

<code>close()</code>	Закрывает окно.
<code>reset_parent(parent)</code>	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
<code>show([type])</code>	Показывает окно в соответствие с приведенным типом.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

`close()`

Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property position: QRect

Размер и положение окна.

reset_parent(parent: QWidget)

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры

parent – Окно - новый родитель

Пример встраивания окно карты в диалог:

```
table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
self.layout.addWidget(view.widget)
view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()
```

show(type: `int` = `SHOW_NORMAL`)

Показывает окно в соответствие с приведенным типом.

Таблица 1: Допустимые значения:

Константа	Значение	Описание
<code>SHOW_NORMAL</code>		Обычный показ окна (по умолчанию).
<code>SHOW_MINIMIZED</code>		Показ окна в режиме минимизации.
<code>SHOW_MAXIMIZED</code>		Показ окна в режиме распахивания.

property show_type: `int`

Возвращает тип состояния окна.

Подробнее см. `show()`

property title: `str`

Заголовок окна просмотра.

property widget: `QWidget`

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.8 TableView - Таблица просмотра атрибутивной информации

class axipy.TableView

Базовые классы: `View`

Таблица просмотра атрибутивной информации.

Для создания экземпляра необходимо использовать `axipy.ViewManager.create_tableview()` через экземпляр `axipy.view_manager`.

Свойства:

<code>data_object</code>	Таблица, на основании которой создается данное окно просмотра.
<code>position</code>	Размер и положение окна.
<code>show_type</code>	Возвращает тип состояния окна.
<code>table_view</code>	Ссылка на объект таблицы просмотра.
<code>title</code>	Заголовок окна просмотра.
<code>widget</code>	Виджет, соответствующий содержимому окна.

Методы:

<code>close()</code>	Закрывает окно.
<code>reset_parent(parent)</code>	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
<code>show([type])</code>	Показывает окно в соответствие с приведенным типом.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`close()`
Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property data_object: Table

Таблица, на основании которой создается данное окно просмотра.

Результат
Таблица.

property position: QRect

Размер и положение окна.

reset_parent(parent: QWidget)

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры
parent – Окно - новый родитель

Пример встраивания окна карты в диалог:

```

table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
        self.layout.addWidget(view.widget)
        view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()

```

show(type: `int` = `SHOW_NORMAL`)

Показывает окно в соответствие с приведенным типом.

Таблица 2: Допустимые значения:

Константа	Значение	Описание
<code>SHOW_NORMAL</code>		Обычный показ окна (по умолчанию).
<code>SHOW_MINIMIZED</code>		Показ окна в режиме минимизации.
<code>SHOW_MAXIMIZED</code>		Показ окна в режиме распахивания.

property show_type: `int`

Возвращает тип состояния окна.

Подробнее см. `show()`

property table_view: `QTableView`

Ссылка на объект таблицы просмотра.

Пример установки сортировки для таблицы в текущем окне по второй колонке по возрастанию:

```

if isinstance(view_manager.active, TableView):
    view_manager.active.table_view.sortByColumn(2, Qt.AscendingOrder)

```

property title: `str`

Заголовок окна просмотра.

property widget: `QWidget`

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.9 DrawableView - Базовый класс с поддержкой визуального редактирования геометрий

class axipy.DrawableView

Базовые классы: [View](#)

Базовый класс для визуализации геометрических данных.

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
device_to_scene_transform	Объект трансформации из координат окна в координаты карты.
is_modified	Есть ли изменения в окне.
position	Размер и положение окна.
scene_to_device_transform	Объект трансформации из координат карты в координаты окна.
show_type	Возвращает тип состояния окна.
snap_mode	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
title	Заголовок окна просмотра.
widget	Виджет, соответствующий содержимому окна.

Методы:

close()	Закрывает окно.
offset(dx, dy)	Производит сдвиг окна карты или отчета.
redo()	Производит откат на один шаг вперед.
reset_parent(parent)	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
scale_with_center(scale, center)	Установка нового центра с заданным масштабированием.
show([type])	Показывает окно в соответствие с приведенным типом.
undo()	Производит откат на один шаг назад.

Сигналы:

scene_changed	Сигнал об изменении контента окна.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`

Оператор равенства (==).

`__repr__() → str`

Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`

Неформальное строковое представление объекта при вызове `str` или `print()`.

property can_redo: bool

Возможен ли откат на один шаг вперед.

property can_undo: bool

Возможен ли откат на один шаг назад.

`close()`

Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property device_to_scene_transform: QTransform

Объект трансформации из координат окна в координаты карты.

Результат

Объект трансформации.

property is_modified: bool

Есть ли изменения в окне.

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

`redo()`

Производит откат на один шаг вперед.

При этом возвращается состояние до последней отмены.

reset_parent(parent: [QWidget](#))

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры

parent – Окно - новый родитель

Пример встраивания окно карты в диалог:

```
table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
        self.layout.addWidget(view.widget)
        view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()
```

scale_with_center(scale: [float](#), center: [Pnt](#))

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: [Signal](#)

Сигнал об изменении контента окна.

Тип результата

[Signal\[\]](#)

property scene_to_device_transform: [QTransform](#)

Объект трансформации из координат карты в координаты окна.

Результат

Объект трансформации.

show(type: [int](#) = [SHOW_NORMAL](#))

Показывает окно в соответствие с приведенным типом.

Таблица 3: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL		Обычный показ окна (по умолчанию).
SHOW_MINIMIZED		Показ окна в режиме минимизации.
SHOW_MAXIMIZED		Показ окна в режиме распахивания.

property show_type: int

Возвращает тип состояния окна.

Подробнее см. `show()`

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.10 MapView - Окно просмотра карты

class axipy.MapView

Базовые классы: `DrawableView`

Окно просмотра карты. Используется для проведения различных манипуляций с картой. Для создания экземпляра необходимо использовать `axipy.ViewManager.create_mapview()` через экземпляр `view_manager` (пример см. ниже).

Примечание: При создании „MapView“ посредством `axipy.ViewManager.create_mapview()` производится клонирование экземпляра карты `axipy.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 уместается 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`.

К параметрам самой карты можно получить доступ через свойство `map`. Единицы измерения координат (`unit`) также берутся из наиболее подходящей СК, но при желании они могут быть изменены. К примеру, вместо метров могут быть установлены километры.

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.Geometry.affine_transform()`.

```
# Откроем таблицу, создадим на ее базе слой и добавим в карту
>>> import axipy
>>> table_world = axipy.provider_manager.openfile('world.tab')
>>> world = Layer.create(table_world)
>>> map_ = Map([world])
# Для полученной карты создадим окно просмотра
>>> mapview = axipy.view_manager.create_mapview(map_)
# Выведем полученные параметры отображения
>>> print('Прямоугольник экрана:', mapview.device_rect)
Прямоугольник экрана: (0.0 0.0) (300.0 200.0)
>>> print('Прямоугольник карты:', mapview.scene_rect)
Прямоугольник карты: (-16194966.287183324 -8621185.324024437) (16789976.633236416,
↪8326222.646170927)
# Установим ширину карты и ее центр
>>> mapview.center = (1000000, 1000000)
>>> mapview.set_zoom(10e6)

# Создадим геометрический объект полигон в координатах экрана и преобразуем его в ↵
↪СК карты
>>> poly_device = axipy.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.to_wkt())
WKT: POLYGON ((-5199985.31371008 -147481.338926755, -5199985.31371008 -4384333.
↪3314756, 297505.173026545 -4384333.3314756, 297505.173026545 -147481.338926755, ↵
↪-5199985.31371008 -147481.338926755))
```

Свойства:

<code>calc_mode</code>	Устанавливает или возвращает режим расчета для карты.
<code>can_redo</code>	Возможен ли откат на один шаг вперед.
<code>can_undo</code>	Возможен ли откат на один шаг назад.
<code>center</code>	Центр окна карты.
<code>clip</code>	Геометрия обрезки карты.
<code>coordsystem</code>	Система координат карты.
<code>coordsystem_visual</code>	Система координат карты с учетом поправки цены градуса по широте.
<code>device_rect</code>	Видимая область в координатах окна (пиксели).
<code>device_to_scene_transform</code>	Объект трансформации из координат окна в координаты карты.
<code>editable_layer</code>	Редактируемый слой на карте.
<code>is_modified</code>	Есть ли изменения в окне.
<code>map</code>	Объект карты.
<code>position</code>	Размер и положение окна.
<code>scale</code>	Масштаб карты.
<code>scene_rect</code>	Видимая область в координатах карты (в единицах измерения СК).
<code>scene_to_device_transform</code>	Объект трансформации из координат карты в координаты окна.
<code>selected_layer</code>	Выделенный слой на карте.
<code>show_type</code>	Возвращает тип состояния окна.
<code>snap_mode</code>	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
<code>title</code>	Заголовок окна просмотра.
<code>unit</code>	Единицы измерения координат карты.
<code>widget</code>	Виджет, соответствующий содержимому окна.

Методы:

<code>close()</code>	Закрывает окно.
<code>offset(dx, dy)</code>	Производит сдвиг окна карты или отчета.
<code>redo()</code>	Производит откат на один шаг вперед.
<code>reset_parent(parent)</code>	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
<code>scale_with_center(scale, center)</code>	Установка нового центра с заданным масштабированием.
<code>set_zoom(zoom[, unit])</code>	Задание ширины окна карты.
<code>set_zoom_and_center(zoom, center[, unit])</code>	Задаёт новый центр и ширину окна карты.
<code>show([type])</code>	Показывает окно в соответствии с приведенным типом.
<code>show_all()</code>	Полностью показывает все слои карты.
<code>show_selection()</code>	Перемещает карту к группе выделенных объектов, максимально увеличивая масштаб, но так, чтобы все объекты попадали.
<code>undo()</code>	Производит откат на один шаг назад.
<code>zoom([unit])</code>	Ширина окна карты.

Сигналы:

<code>coordsystem_changed</code>	Сигнал о том, что система координат изменилась.
<code>editable_layer_changed</code>	Сигнал о том, что редактируемый слой сменился.
<code>mouse_moved</code>	Сигнал при смещении курсора мыши.
<code>scene_changed</code>	Сигнал об изменении контента окна.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (<code>==</code>).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (`==`).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property calc_mode: CalcMode

Устанавливает или возвращает режим расчета для карты.

Пример установки для активной карты:

```
if isinstance(view_manager.active, MapView):
    view_manager.active.calc_mode = CalcMode.ELLIPSOID
```

property can_redo: `bool`

Возможен ли откат на один шаг вперед.

property can_undo: `bool`

Возможен ли откат на один шаг назад.

property center: `Pnt`

Центр окна карты.

property clip: `ClipGeometry`

Геометрия обрезки карты. Устанавливается геометрия, в рамках которой будет отрисована карта. За пределами отрисовка производиться не будет. Обрабатываются только площадные объекты. Так-же допустимо устанавливать коллекции.

Пример установки обрезки по заданной геометрии:

```
rectangle = axipy.Rectangle(axipy.Rect(-50, -50, 50, 50))
axipy.view_manager.active.clip.geometry = rectangle
```

Если установить None, то режим сбрасывается:

```
axipy.view_manager.active.clip.geometry = None
```

Если необходимо сохранить объект, то можно просто отключить режим:

```
axipy.view_manager.active.clip.status = False
```

close()

Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property coordsystem: `CoordSystem`

Система координат карты.

property coordsystem_changed: `Signal`

Сигнал о том, что система координат изменилась.

Пример::

```
>>> import axipy
>>> layer_world = axipy.Layer.create(table_world)
>>> map_ = axipy.Map([layer_world])
>>> mapview = axipy.view_manager.create_mapview(map_)
>>> mapview.coordsystem_changed.connect(lambda: print('СК была изменена'))
>>> csLL = axipy.CoordSystem.from_prj("1, 104")
>>> mapview.coordsystem = csLL
СК была изменена
```

property coordsystem_visual: CoordSystem

Система координат карты с учетом поправки цены градуса по широте.

Отличается от `coordsystem` только лишь в случае, когда основная система координат - это широта/долгота и широта имеет ненулевое значение. При этом в диапазоне широты (-70...70) градусов вводится поправочный коэффициент, растягивающий изображение по широте и равный $1/\cos(y)$.

property device_rect: Rect

Видимая область в координатах окна (пиксели).

Результат

Прямоугольник в координатах окна.

property device_to_scene_transform: QTransform

Объект трансформации из координат окна в координаты карты.

Результат

Объект трансформации.

property editable_layer: Optional[VectorLayer]

Редактируемый слой на карте.

Результат

Редактируемый слой. Если не определен, возвращает None.

property editable_layer_changed: Signal

Сигнал о том, что редактируемый слой сменился.

property is_modified: bool

Есть ли изменения в окне.

property map: Map

Объект карты.

Результат

Карта.

property mouse_moved: Signal

Сигнал при смещении курсора мыши. Возвращает значения в СК карты.

Тип результата

Signal[float, float]

Пример:

```
mapview.mouse_moved.connect(lambda x,y: print('Coords: {} {}'.format(x, y)))

>> Coords: -5732500.0 958500.0
>> Coords: -5621900.0 847900.0
```

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

redo()

Производит откат на один шаг вперед.

При этом возвращается состояние до последней отмены.

reset_parent(parent: QWidget)

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры

parent – Окно - новый родитель

Пример встраивания окно карты в диалог:

```
table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
        self.layout.addWidget(view.widget)
        view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()
```

property scale: float

Масштаб карты.

scale_with_center(scale: float, center: Pnt)

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: Signal

Сигнал об изменении контента окна.

Тип результата

Signal[]

property scene_rect: **Rect**

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

Результат

Прямоугольник в координатах карты.

property scene_to_device_transform: **QTransform**

Объект трансформации из координат карты в координаты окна.

Результат

Объект трансформации.

property selected_layer: **Optional[VectorLayer]**

Выделенный слой на карте.

Результат

Выделенный слой. Если выделение отсутствует, возвращает None.

set_zoom(zoom: **float**, unit: **Optional[LinearUnit]** = None)

Задание ширины окна карты.

Параметры

- **zoom** – Значение ширины карты.
- **unit** – Единицы измерения. Если не заданы, берутся текущие для карты.

set_zoom_and_center(zoom: **float**, center: **Pnt**, unit: **Optional[LinearUnit]** = None)

Задаёт новый центр и ширину окна карты.

Параметры

- **zoom** – Значение ширины карты.
- **center** – Центр карты.
- **unit** – Единицы измерения. Если не заданы, берутся текущие для карты.

show(type: **int** = SHOW_NORMAL)

Показывает окно в соответствии с приведенным типом.

Таблица 4: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL		Обычный показ окна (по умолчанию).
SHOW_MINIMIZED		Показ окна в режиме минимизации.
SHOW_MAXIMIZED		Показ окна в режиме распахивания.

show_all()

Полностью показывает все слои карты.

show_selection()

Перемещает карту к группе выделенных объектов, максимально увеличивая масштаб, но так, чтобы все объекты попадали.

property show_type: **int**

Возвращает тип состояния окна.

Подробнее см. [show\(\)](#)

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property unit: LinearUnit

Единицы измерения координат карты.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

zoom(unit: Optional[LinearUnit] = None) → float

Ширина окна карты.

Параметры

unit – Единицы измерения. Если не заданы, берутся текущие для карты.

24.11 ReportView - Окно просмотра отчета

class axipy.ReportView

Базовые классы: `DrawableView`

Окно с планом отчета. Для создания экземпляра необходимо использовать `axipy.ViewManager.create_reportview()`. через экземпляр `axipy.view_manager`. Параметры отчета `axipy.Report` можно получить через свойство `ReportView.report()`.

Пример создания отчета:

```
reportview = view_manager.create_reportview()

# Добавим полигон
geomItem = GeometryReportItem()
geomItem.geometry = Polygon((10,10), (10, 100), (100, 100), (10, 10))
geomItem.style = PolygonStyle(45, Qt.red)
reportview.report.items.add(geomItem)

# Установим текущий масштаб
reportview.view_scale = 33
```

Свойства:

can_redo	Возможен ли откат на один шаг вперед.
can_undo	Возможен ли откат на один шаг назад.
device_to_scene_transform	Объект трансформации из координат окна в координаты карты.
is_modified	Есть ли изменения в окне.
mesh_size	Размер ячейки сетки.
position	Размер и положение окна.
report	Объект отчета.
scene_to_device_transform	Объект трансформации из координат карты в координаты окна.
show_borders	Показывать границы страниц.
show_elements_size	Показывать размер элементов.
show_mesh	Показывать сетку привязки.
show_ruler	Показывать линейку по краям.
show_type	Возвращает тип состояния окна.
snap_mode	Включает режим привязки координат при редактировании геометрии в окне карты или отчета.
snap_to_guidelines	Включение режима притяжения элементов отчета к направляющим.
snap_to_mesh	Включение режима притяжения элементов отчета к узлам сетки.
title	Заголовок окна просмотра.
view_scale	Текущий масштаб.
widget	Виджет, соответствующий содержимому окна.
x_guidelines	Вертикальные направляющие.
y_guidelines	Горизонтальные направляющие.

Методы:

<code>clear_guidelines()</code>	Удаляет все направляющие.
<code>clear_selected_guidelines()</code>	Очищает выбранные направляющие.
<code>close()</code>	Закрывает окно.
<code>fill_on_pages()</code>	Наиболее эффективно заполняет пространство отчета масштабированием его элементов.
<code>get_printer()</code>	Ссылка на используемый текущий принтер.
<code>offset(dx, dy)</code>	Производит сдвиг окна карты или отчета.
<code>redo()</code>	Производит откат на один шаг вперед.
<code>reset_parent(parent)</code>	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
<code>save_template(file_name)</code>	Сохраняет отчет в виде шаблона как файл *.tmpl
<code>scale_with_center(scale, center)</code>	Установка нового центра с заданным масштабированием.
<code>set_printer(printer)</code>	Устанавливает для отчета объект <code>PySide2.QtPrintSupport.QPrinter</code>
<code>show([type])</code>	Показывает окно в соответствии с приведенным типом.
<code>undo()</code>	Производит откат на один шаг назад.

Сигналы:

<code>mouse_moved</code>	Сигнал при смещении курсора мыши.
<code>scene_changed</code>	Сигнал об изменении контента окна.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

property can_redo: bool
Возможен ли откат на один шаг вперед.

property can_undo: bool
Возможен ли откат на один шаг назад.

clear_guidelines()

Удаляет все направляющие.

clear_selected_guidelines()

Очищает выбранные направляющие.

close()

Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property device_to_scene_transform: QTransform

Объект трансформации из координат окна в координаты карты.

Результат

Объект трансформации.

fill_on_pages()

Наиболее эффективно заполняет пространство отчета масштабированием его элементов.

get_printer() → QPrinterСсылка на используемый текущий принтер. Для того, чтобы изменить настройки, нужно запросить существующий объект, поменять необходимые значения и снова назначить посредством `ReportView.set_printer()`. Или же установить другой объект `PySide2.QtPrintSupport.QPrinter`.

Список 2: Пример смены свойств у текущего окна отчета

```
from PySide2.QtGui import QPageLayout, QPageSize
import axipy
# Получение текущего окна отчета
reportview = axipy.view_manager.active
if isinstance(reportview, axipy.ReportView):
    # Получение текущего принтера
    printer = reportview.get_printer()
    # Изменение размера страницы
    printer.setPageSize(QPageSize(QPageSize.A3))
    # Изменение ориентации страницы
    printer.setPageOrientation(QPageLayout.Landscape)
    # Установление нового значения
    reportview.set_printer(printer)
```

property is_modified: bool

Есть ли изменения в окне.

property mesh_size: float

Размер ячейки сетки.

property mouse_moved: Signal

Сигнал при смещении курсора мыши. Возвращает значения в координатах отчета.

Тип результату

Signal[float, float]

Пример:

```
reportview.mouse_moved.connect(lambda x,y: print('Coords: {} {}'.format(x,
↵y)))
```

offset(dx: float, dy: float)

Производит сдвиг окна карты или отчета. Особенностью является то, что при этом сохраняется прежний центр (актуально для карты).

Параметры

- **dx** – Смещение по горизонтали в координатах экрана (пикселях)
- **dy** – Смещение по вертикали в координатах экрана (пикселях)

property position: QRect

Размер и положение окна.

redo()

Производит откат на один шаг вперед.

При этом возвращается состояние до последней отмены.

property report: Report

Объект отчета.

Результат

Отчет.

reset_parent(parent: QWidget)

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры

parent – Окно - новый родитель

Пример встраивания окна карты в диалог:

```
table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
        self.layout.addWidget(view.widget)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()
```

save_template(file_name: **str**)

Сохраняет отчет в виде шаблона как файл *.tmpl

Параметры

file_name – Наименование выходного файла.

scale_with_center(scale: **float**, center: **Pnt**)

Установка нового центра с заданным масштабированием.

Параметры

- **scale** – Коэффициент масштабирования по отношению к текущему.
- **center** – Устанавливаемый центр.

property scene_changed: **Signal**

Сигнал об изменении контента окна.

Тип результата

Signal[]

property scene_to_device_transform: **QTransform**

Объект трансформации из координат карты в координаты окна.

Результат

Объект трансформации.

set_printer(printer: **QPrinter**)

Устанавливает для отчета объект **PySide2.QtPrintSupport.QPrinter**

Параметры

printer – Новое значение принтера или измененное запрошенное ранее через **ReportView.get_printer()**

show(type: **int** = **SHOW_NORMAL**)

Показывает окно в соответствии с приведенным типом.

Таблица 5: Допустимые значения:

Константа	Значение	Описание
SHOW_NORMAL		Обычный показ окна (по умолчанию).
SHOW_MINIMIZED		Показ окна в режиме минимизации.
SHOW_MAXIMIZED		Показ окна в режиме распахивания.

property show_borders: **bool**

Показывать границы страниц.

property show_elements_size: **bool**

Показывать размер элементов.

property show_mesh: bool

Показывать сетку привязки.

property show_ruler: bool

Показывать линейку по краям.

property show_type: int

Возвращает тип состояния окна.

Подробнее см. `show()`

property snap_mode: bool

Включает режим привязки координат при редактировании геометрии в окне карты или отчета.

property snap_to_guidelines: bool

Включение режима притяжения элементов отчета к направляющим.

property snap_to_mesh: bool

Включение режима притяжения элементов отчета к узлам сетки.

property title: str

Заголовок окна просмотра.

undo()

Производит откат на один шаг назад.

property view_scale: float

Текущий масштаб.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

property x_guidelines: XGuidelines

Вертикальные направляющие. Значения содержатся в единицах измерения отчета.

Рассмотрим на примере:

```
# Добавление вертикальной направляющей
reportview.x_guidelines.append(20)
# Изменение значения направляющей по индексу
reportview.x_guidelines[0] = 80
# Удаление всех направляющих.
reportview.clear_guidelines()
```

property y_guidelines: YGuidelines

Горизонтальные направляющие.

Работа с ними производится по аналогии с вертикальными направляющими.

24.12 LegendView - Окно просмотра легенд карты

class axipy.LegendView

Базовые классы: `View`

Легенда для карты. Для создания экземпляра необходимо использовать `axipy.ViewManager.create_legendview()` через экземпляр `view_manager`. В качестве параметра передается открытое ранее окно с картой:

```
legendView = view_manager.create_legendview(map_view)
```

Список легенд доступен через свойство `legends`:

```
for legend in legendView.legends:
    print(legend.caption)
```

Состав может меняться посредством вызова соответствующих методов свойства `legends`.

Добавление легенды для слоя карты:

```
legend = Legend(map_view.map.layers[0])
legend.caption = 'Легенда слоя'
legendView.legends.append(legend)
legendView.arrange()
```

Доступ к элементу по индексу. Поменяем описание четвертого оформления у первой легенды `axipy.Legend` окна:

```
legend = legendView.legends[1]
item = legend.items[3]
item.title = 'Описание'
legend.items[3] = item
```

Удаление первой легенды из окна:

```
legendView.legends.remove(0)
```

Свойства:

<code>legends</code>	Перечень добавленных в окно легенд.
<code>position</code>	Размер и положение окна.
<code>show_type</code>	Возвращает тип состояния окна.
<code>title</code>	Заголовок окна просмотра.
<code>widget</code>	Виджет, соответствующий содержанию окна.

Методы:

<code>arrange()</code>	Упорядочивает легенды с целью устранения наложений легенд друг на друга.
<code>close()</code>	Закрывает окно.
<code>reset_parent(parent)</code>	Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его.
<code>show([type])</code>	Показывает окно в соответствие с приведенным типом.

Специальные методы:

<code>__eq__(other)</code>	Оператор равенства (==).
<code>__repr__()</code>	Официальное строковое представление объекта при вызове <code>repr()</code> .
<code>__str__()</code>	Неформальное строковое представление объекта при вызове <code>str</code> или <code>print()</code> .

`__eq__(other: object) → bool`
Оператор равенства (==).

`__repr__() → str`
Официальное строковое представление объекта при вызове `repr()`.

`__str__() → str`
Неформальное строковое представление объекта при вызове `str` или `print()`.

`arrange()`
Упорядочивает легенды с целью устранения наложений легенд друг на друга.

`close()`
Закрывает окно.

Примечание: Если при закрытии окна выдается запрос о сохранении данных перед закрытием и необходимо изменить поведение, то это можно сделать через соответствующие типу окна настройки или же через свойство `axipy.CurrentSettings.SilentCloseWidget`

property legends: ListLegend

Перечень добавленных в окно легенд.

property position: QRect

Размер и положение окна.

reset_parent(parent: QWidget)

Сбрасывает окно контейнера для карты или таблицы просмотра, если он существует и закрывает его. Это требуется когда окно карты или таблица просмотра необходимо встроить в другое окно.

Параметры

parent – Окно - новый родитель

Пример встраивания окно карты в диалог:

```

table = provider_manager.openfile('world.tab')
layer = Layer.create(table)
m = Map([layer])
view = view_manager.create_mapview(m)

# Диалог, в который будет встраиваться
class MyDialog(QDialog):

    def __init__(self, parent ):
        super().__init__(parent)
        # Инициализируем менеджер компоновки
        self.layout = QGridLayout()
        self.setLayout(self.layout)

    def set_mapview(self, view):
        # Устанавливаем окно карты как контент данного диалога
        view.reset_parent(self)
        self.layout.addWidget(view.widget)
        view.show()

dialog = MyDialog(view_manager.global_parent)
dialog.resize(500,300)
dialog.set_mapview(view)
dialog.show()

```

show(type: `int` = `SHOW_NORMAL`)

Показывает окно в соответствие с приведенным типом.

Таблица 6: Допустимые значения:

Константа	Значение	Описание
<code>SHOW_NORMAL</code>		Обычный показ окна (по умолчанию).
<code>SHOW_MINIMIZED</code>		Показ окна в режиме минимизации.
<code>SHOW_MAXIMIZED</code>		Показ окна в режиме распаивания.

property show_type: `int`

Возвращает тип состояния окна.

Подробнее см. `show()`

property title: `str`

Заголовок окна просмотра.

property widget: `QWidget`

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.13 ListLegend - Список легенд

class axipy.ListLegend

Базовые классы: `object`

Список добавленных в окно легенд.

Специальные методы:

<code>__getitem__(idx)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__getitem__(idx: int) → Legend`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

24.14 SelectionManager - Доступ к выделенным объектам

class axipy.SelectionManager

Класс доступа к выделенным объектам.

Примечание: Создание `axipy.SelectionManager` не требуется, используйте объект `axipy.selection_manager`.

Свойства:

<code>count</code>	Размер выделения, то есть количество выделенных записей (количество элементов в списке идентификаторов).
<code>ids</code>	Список идентификаторов выделенных записей.
<code>table</code>	Таблица, являющаяся источником текущего выделения.

Методы:

<code>add(table, ids)</code>	Добавляет выборку записи таблицы по их идентификаторам.
<code>clear()</code>	Очищает выборку.
<code>get_as_cursor()</code>	Возвращает выборку в виде итератора по записям.
<code>get_as_table()</code>	Возвращает выборку в виде таблицы.
<code>remove(table, ids)</code>	Удаляет из выборки записи таблицы по их идентификаторам.
<code>set(table, ids)</code>	Создает выборку из записей таблицы по их идентификаторам.
<code>set_all(table)</code>	Создает выборку по всем записям таблицы.

Сигналы:

<code>changed</code>	Выделение было изменено.

Специальные методы:

<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__len__() → int`

Длина объекта (функция `len()`).

add(table: `Table`, ids: `Union[List[int], int]`)

Добавляет выборку записи таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

Пример установки выборки из запроса:

```
qry = data_manager.query('select * from world where Столица like "М%")
for feature in qry.items():
    selection_manager.add(qry, feature.id)
```

property changed: `Signal`

Выделение было изменено.

Тип результата

`Signal[]`

clear()

Очищает выборку.

property count: `int`

Размер выделения, то есть количество выделенных записей (количество элементов в списке идентификаторов).

get_as_cursor() → `Iterator[Feature]`

Возвращает выборку в виде итератора по записям.

Пример:

```
for f in selection_manager.get_as_cursor():
    print('Feature id={}. Страна={}'.format(f.id, f['Страна']))
```

Предупреждение: Не рекомендуется, начиная с версии 3.5: Используйте `axipy.DataManager.selection`.

get_as_table() → `Optional[Table]`

Возвращает выборку в виде таблицы.

Результат

Таблица или `None`, если выборки нет.

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_manager.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']))
```

Предупреждение: Не рекомендуется, начиная с версии 3.5: Используйте `axipy.DataManager.selection`.

property ids: `List[int]`

Список идентификаторов выделенных записей.

remove(table: `Table`, ids: `Union[List[int], int]`)

Удаляет из выборки записи таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

set(table: `Table`, ids: `Union[List[int], int]`)

Создает выборку из записей таблицы по их идентификаторам.

Параметры

- **table** – Таблица.
- **ids** – Идентификаторы записей.

Пример выбора всех объектов слоя:

```
table = data_manager.find('world')
if table:
    selection_manager.set(table, [f.id for f in table.items()])
```

set_all(table: Table)

Создает выборку по всем записям таблицы.

Параметры

table - Таблица.

Пример выбора всех объектов слоя:

```
table = data_manager.find('world')
if table:
    selection_manager.set_all(table)
```

property table: Optional[Table]

Таблица, являющаяся источником текущего выделения.

Если ничего не выделено, то None.

24.15 ViewManager - Менеджер содержимого окон

class ахіру.ViewManager

Менеджер содержимого окон.

Примечание: Создание `ахіру.ViewManager` не требуется, используйте объект `ахіру.view_manager`.

Список 3: Пример использования

```
table = provider_manager.openfile(filepath)
m = Map([table])
view_manager.create_mapview(m)
```

Свойства:

<code>active</code>	Текущее активное окно.
<code>count</code>	Количество окон.
<code>data_managers</code>	Список виджетов управления списком открытых данных.
<code>global_parent</code>	Этот параметр можно использовать как родительское окно 'parent' при открытии диалогов.
<code>layer_controls</code>	Список виджетов управления слоями карты.
<code>legendviews</code>	Список всех окон с легендами.
<code>mapviews</code>	Список всех окон с картами.
<code>reportviews</code>	Список всех окон с отчетами.
<code>tableviews</code>	Список всех окон с таблицами просмотра.
<code>views</code>	Список всех известных окон.

Методы:

<code>activate(view)</code>	Делает заданное окно активным.
<code>add_to_current_mapview(value)</code>	Добавляет слой в текущее окно.
<code>close(view)</code>	Закрывает окно.
<code>close_all()</code>	Закрывает все окна.
<code>create_legendview(mapview)</code>	Создает окно легенды для карты.
<code>create_mapview(map)</code>	Создает окно из для переданного объекта карты.
<code>create_reportview([report])</code>	Создает окно с планом отчета.
<code>create_tableview(table)</code>	Создает окно в виде табличного представления из объекта данных.
<code>create_view(value)</code>	Создает окно из для переданного объекта соответствующего типа.

Сигналы:

<code>active_changed</code>	Активное окно изменилось.
<code>count_changed</code>	Количество окон изменилось.
<code>mainwindow_activated</code>	Возникает когда главное окно приложения инициализировано и активировано.

Специальные методы:

<code>__contains__(key)</code>	Проверяет существование окна с заданным именем.
<code>__getitem__(key)</code>	Возвращает существующее окно по имени.
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__contains__(key: str) → bool`

Проверяет существование окна с заданным именем.

Параметры

key – Имя окна.

`__getitem__(key: str) → View`

Возвращает существующее окно по имени.

Исключение

KeyError – Окно с заданным именем не найдено.

`__iter__() → Iterator[View]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

activate(view: View)

Делает заданное окно активным.

Параметры

view – Содержимое окна.

property active: `Optional[View]`

Текущее активное окно.

Результат

None, если нет активных окон.

property active_changed: `Signal`

Активное окно изменилось.

Тип результата

`Signal[QWidget]`

add_to_current_mapview(value: `Union[Layer, DataObject]`)

Добавляет слой в текущее окно. Если текущее окно карты отсутствует, создается новое окно.

Параметры

value – Слой или объект данных

close(view: `View`)

Закрывает окно.

Параметры

view – Содержимое окна.

close_all()

Закрывает все окна.

property count: `int`

Количество окон.

property count_changed: `Signal`

Количество окон изменилось.

Тип результата

`Signal[]`

create_legendview(mapview: `MapView`) → `LegendView`

Создает окно легенды для карты.

Параметры

mapview – Окно с картой.

Результат

Окно с легендой.

create_mapview(map: `Union[Map, Layer, DataObject, List[Layer]]`) → `MapView`

Создает окно из для переданного объекта карты.

Параметры

map – Карта, слой или объект данных.

Примечание: Переданная карта копируется.

Результат

Окно карты.

create_reportview(report: [Optional\[Report\]](#) = None) → [ReportView](#)

Создает окно с планом отчета.

Параметры

report – План отчета. Если не передан, то создается по умолчанию.

Результат

Окно отчета.

create_tableview(table: [Table](#)) → [TableView](#)

Создает окно в виде табличного представления из объекта данных.

Параметры

table – Таблица.

Результат

Окно таблицы.

create_view(value: [Union\[Map, Layer, DataObject, Report, MapView\]](#)) → [Union\[MapView, TableView, ReportView, LegendView\]](#)

Создает окно из для переданного объекта соответствующего типа.

Параметры

value – таблица, слой, карта или отчет.

Результат

Окно.

property data_managers: [Tuple\[DataManagerWidget, ...\]](#)

Список виджетов управления списком открытых данных.

property global_parent: [QWidget](#)

Этот параметр можно использовать как родительское окно „parent“ при открытии диалогов. Если не указывать родителя, диалог может отображаться как отдельное окно, независимое от основного окна программы.

Пример:

```
if QMessageBox.question(view_manager.global_parent, 'Вопрос', 'Отменить_
↪действие?') == QMessageBox.Yes:
    pass
```

property layer_controls: [Tuple\[LayerControlWidget, ...\]](#)

Список виджетов управления слоями карты.

property legendviews: [List\[LegendView\]](#)

Список всех окон с легендами.

property mainwindow_activated: [Signal](#)

Возникает когда главное окно приложения инициализировано и активировано. Данное событие может использоваться в плагинах, когда есть необходимость обратиться к главному окну приложения. Но ввиду того, что сами плагины загружаются до инициализации главного окна, данную процедуру можно выполнить используя данное событие (`axipy.AxiomaInterface.window`).

Тип результата

[Signal\[\]](#)

property mapviews: `List[MapView]`

Список всех окон с картами.

Пример:

```
for v in view_manager.mapviews:
    print('Widget:', v.title)

```>>> Widget: Карта: world```
```>>> Widget: Карта: rus_obl```
```

property reportviews: `List[ReportView]`

Список всех окон с отчетами.

property tableviews: `List[TableView]`

Список всех окон с таблицами просмотра.

property views: `List[View]`

Список всех известных окон.

24.16 ActionManager - Менеджер системных действий и инструментов

class `axipy.ActionManager`

Менеджер системных действий и инструментов. Класс является словарем, доступным только для чтения (`collections.abc.Mapping`), где ключи это идентификаторы действий, а значения это объекты класса `PySide2.QtWidgets.QAction`. Поддерживает обращение по ключу.

Примечание: Создание `axipy.ActionManager` не требуется, используйте объект `axipy.action_manager`.

Вывод перечня доступных действий:

```
print(action_manager.keys())
```

Получение действия по его наименованию:

```
if 'Select' in action_manager.keys():
    action = action_manager['Select']
```

Добавление действия в основную панель:

```
if 'Select' in action_manager.keys():
    position = Position('Основные', 'Команды')
    button = SystemActionButton('Select')
    position.add(button)
```

См.также:

`MainWindow.quick_toolbar`

Свойства:

<code>icons</code>	Возвращает словарь, доступный только для чтения (<code>collections.abc.Mapping</code>), где ключи это идентификаторы иконок, а значения это объекты класса <code>PySide2.QtGui.QIcon</code> .

Методы:

<code>activate(name)</code>	Делает активным инструмент по его идентификатору если это возможно.
<code>activate_selection_tool()</code>	Активирует инструмент "Выбор" (инструмент по умолчанию).
<code>find_by_text(text)</code>	Найти действие по его тексту.
<code>get(key[, default_value])</code>	Возвращает значение по ключу.
<code>items()</code>	Возвращает набор кортежей ключ-значение, где ключи это идентификаторы действий, а значения это объекты класса <code>PySide2.QtWidgets.QAction</code> .
<code>keys()</code>	Возвращает набор ключей, где ключи это идентификаторы действий.
<code>values()</code>	Возвращает коллекцию значений, где значения это объекты класса <code>PySide2.QtWidgets.QAction</code> .

Специальные методы:

<code>__getitem__(key)</code>	Доступ к элементу по индексу (<code>obj[i]</code>).
<code>__iter__()</code>	Возвращает итератор по объекту (функция <code>iter()</code>).
<code>__len__()</code>	Длина объекта (функция <code>len()</code>).

`__getitem__(key: str) → QAction`

Доступ к элементу по индексу (`obj[i]`).

`__iter__() → Iterator[str]`

Возвращает итератор по объекту (функция `iter()`).

`__len__() → int`

Длина объекта (функция `len()`).

activate(name: str)

Делает активным инструмент по его идентификатору если это возможно. Если действие не найдено, генерируется исключение. Если действие недоступно в настоящий момент (неактивно), установка игнорируется.

Параметры

name – Идентификатор действия

Исключение

ValueError – Если действие не найдено.

Активация инструмента „Сдвиг“:

```
axipy.action_manager.activate('Pan')
```

Вызов диалога „Стиль символа“:

```
axipy.action_manager.activate('SymbolStyle')
```

activate_selection_tool()

Активирует инструмент «Выбор» (инструмент по умолчанию).

find_by_text(text: str) → Optional[QAction]

Найти действие по его тексту.

get(key: str, default_value: Optional[Any] = None) → Optional[QAction]

Возвращает значение по ключу.

property icons: Mapping[str, QIcon]

Возвращает словарь, доступный только для чтения (`collections.abc.Mapping`), где ключи это идентификаторы иконок, а значения это объекты класса `PySide2.QtGui.QIcon`.

items() → ItemsView[str, QAction]

Возвращает набор кортежей ключ-значение, где ключи это идентификаторы действий, а значения это объекты класса `PySide2.QtWidgets.QAction`.

keys() → KeysView[str]

Возвращает набор ключей, где ключи это идентификаторы действий.

values() → ValuesView[QAction]

Возвращает коллекцию значений, где значения это объекты класса `PySide2.QtWidgets.QAction`.

24.17 Workspace - Рабочее пространство

class axipy.Workspace

Сохранение рабочего пространства в файл рабочего набора *.mws.

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 каталог `axipy.DataManager`, доступный через переменную `axipy.data_manager`. А окна с наполнением добавляются в менеджер окон `axipy.ViewManager`, доступный через переменную `view_manager`.

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()`.

Пример чтения данных в рамках отдельного приложения:

```
from axipy import init_axioma, data_manager, view_manager, Workspace, open_file_
    ↳ dialog

# инициализация ядра
app = init_axioma()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
print('Before: tables({}), views({})'.format(len(data_manager), len(view_
↪manager)))
# Чтение рабочего набора
file_name = open_file_dialog("Рабочий набор (*.mws)")
Workspace().load_file(file_name)
print('After: tables({}), views({})'.format(len(data_manager), len(view_manager)))
# Если есть хотя бы одно окно с картой, показываем его
if len(view_manager.mapviews):
    mapview = view_manager.mapviews[0]
    mapview.show()
app.exec_()

# Результат:
# Before: tables(0), views(0)
# After: tables(5), views(3) # В зависимости от рабочего набора, числа могут
↪отличаться
```

Пример записи рабочего пространства:

```
from pathlib import Path

from PySide2.QtCore import QStandardPaths

from axipy import Workspace

path = Path(QStandardPaths.writableLocation(QStandardPaths.DocumentsLocation))
path = path / "ws_out.mws"
Workspace().save_file(path)
print("Файл рабочего набора сохранен.", path)

# Результат:
# Файл рабочего набора сохранен. C:\Users\User\Documents\ws_out.mws
```

Методы:

<code>load_file(file_name[, suppress_non_exist])</code>	Читает из файла рабочего пространства и заносит данные в текущее окружение.
<code>load_string(workspace_str[, suppress_non_exist])</code>	Читает из строки описание рабочего пространства и заносит данные в текущее окружение.
<code>save_file(file_name[, overwrite])</code>	Сохраняет текущее состояние в файл рабочего пространства.
<code>save_string()</code>	Сохраняет текущее состояние рабочего пространства в виде строки.

load_file(file_name: Union[str, Path], suppress_non_exist: bool = False)

Читает из файла рабочего пространства и заносит данные в текущее окружение.

Параметры

- **file_name** – Имя входного файла.
- **suppress_non_exist** – Пропускать ненайденные файлы

load_string(workspace_str: str, suppress_non_exist: bool = False)

Читает из строки описание рабочего пространства и заносит данные в текущее окружение.

Параметры

- **workspace_str** - Строка с контентом рабочего пространства.
- **suppress_non_exist** - Пропускать ненайденные файлы.

save_file(file_name: Union[str, Path], overwrite: bool = False)

Сохраняет текущее состояние в файл рабочего пространства.

Параметры

- **file_name** - Имя выходного файла.
- **overwrite** - Перезаписывать существующий файл. По умолчанию False.

save_string() → str

Сохраняет текущее состояние рабочего пространства в виде строки.

Результат

Строка с контентом рабочего пространства.

24.18 ChooseCoordSystemDialog - Диалог выбора СК

class axipy.ChooseCoordSystemDialog

Диалог выбора координатной системы.

Пример:

```
csMercator = CoordSystem.from_prj('10, 104, 7, 0')
dialog = ChooseCoordSystemDialog(csMercator)
if dialog.exec() == QDialog.Accepted:
    result_cs = dialog.chosenCoordSystem()
    print("Выбрано:", result_cs.title)
```

См.также:

axipy.prompt_coord_system

Конструктор класса:

<code>__init__</code> ([coordsystem])	Конструктор класса.

Методы:

<code>chosenCoordSystem</code> ()	Возвращает выбранную в диалоге систему координат.

`__init__`(coordsystem: Optional[CoordSystem] = None)

Конструктор класса.

Параметры

coordsystem – Система координат по умолчанию. Если не указана, то задается как значение по умолчанию `axipy.CoordSystem.current()`

chosenCoordSystem() → `CoordSystem`

Возвращает выбранную в диалоге систему координат.

24.19 PasswordDialog - Диалог аутентификации пользователя.

class `axipy.PasswordDialog`

Диалог задания или корректировки данных аутентификации пользователя.

Пример:

```
dialog = PasswordDialog()
dialog.address = 'proxy.server'
dialog.user_name = 'user'
dialog.password = 'pass'
if dialog.exec() == QDialog.Accepted:
    print(dialog.user_name, dialog.password)
```

Свойства:

<code>address</code>	Устанавливает или возвращает описание ресурса.
<code>password</code>	Устанавливает или возвращает пароль.
<code>user_name</code>	Устанавливает или возвращает имя пользователя.

property `address`: `str`

Устанавливает или возвращает описание ресурса.

Если значение указано, то в диалоговом окне оно будет присутствовать под именем „Адрес“.

property `password`: `str`

Устанавливает или возвращает пароль.

property `user_name`: `str`

Устанавливает или возвращает имя пользователя.

24.20 BoundingRectDialog - Диалог задания параметров прямоугольника

class `axipy.BoundingRectDialog`

Диалог задания параметров прямоугольника. Например, охвата для таблицы или системы координат.

Пример:

```
dlg = BoundingRectDialog()
dlg.unit = 'км'
dlg.is_square = True
dlg.rect = Rect(-100, -120, 200, 200)
if dlg.exec() == QDialog.Accepted:
    print('>>>>', dlg.rect)
```

См.также:

axipy.prompt_rect

Конструктор класса:

<code>__init__()</code>	Конструктор класса.

Свойства:

<code>is_square</code>	Устанавливает или возвращает контроль соблюдения равенства ширины и высоты.
<code>rect</code>	Устанавливает или возвращает параметры прямоугольника.
<code>unit</code>	Устанавливает или возвращает наименование единиц измерения.

`__init__()`
Конструктор класса.

property is_square: bool

Устанавливает или возвращает контроль соблюдения равенства ширины и высоты.

При этом, после ввода какого-либо значения, производится коррекция таким образом, чтобы в результате получился квадрат.

property rect: Rect

Устанавливает или возвращает параметры прямоугольника.

property unit: str

Устанавливает или возвращает наименование единиц измерения.

24.21 StyleButton - Кнопка выбора стиля

class axipy.StyleButton

Кнопка, отображающая стиль и позволяющая его менять.

Пример добавления кнопки на диалог:

```
from PySide2.QtWidgets import QDialog
from axipy import Style, StyleButton

style = Style.from_mapinfo("Pen (1, 2, 8421504) Brush (2, 255, 0)")
```

(continues on next page)

(продолжение с предыдущей страницы)

```
class Dialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.pb = StyleButton(style, parent=self)
        self.pb.style_changed.connect(self.style_result)
        self.pb.setGeometry(100, 100, 100, 50)

    def style_result(self):
        print("Стиль изменен", self.pb.style)

dialog = Dialog()
dialog.open()
```

Конструктор класса:

<code>__init__</code> ([style, parent])	Конструктор класса.

Свойства:

<code>style</code>	Устанавливает или возвращает стиль кнопки.

Сигналы:

<code>style_changed</code>	Сигнал смены стиля.

`__init__`(style: `Optional[Style]` = None, parent: `Optional[QWidget]` = None)
Конструктор класса.

Параметры

- **style** – Стиль по умолчанию.
- **parent** – Родительский виджет.

property style: `Optional[Style]`

Устанавливает или возвращает стиль кнопки.

property style_changed: `Signal`

Сигнал смены стиля.

24.22 Диалог задания стиля

`axipy.select_style_dialog`(value: `StyleGeometryType`) → `Optional[Style]`

`axipy.select_style_dialog`(value: `Style`) → `Optional[Style]`

`axipy.select_style_dialog`(value: `Geometry`) → `Optional[Style]`

Диалог выбора стиля по геометрии в виде функции. В качестве входных параметров рассматривается или стиль или геометрия, для которого этот стиль выбирается.

Параметры

value – Геометрия, ее тип или стиль. Если задана геометрия, то стиль по умолчанию подставляется как стиль нового объекта для данного типа геометрии.

Результат

Возвращает выбранный в диалоге стиль.

По типу геометрического объекта:

```
style = axipy.select_style_dialog(StyleGeometryType.Linear)
```

Пример задания по геометрии:

```
style = axipy.select_style_dialog(Point(3,3))
```

Пример задания по стилю:

```
style = select_style_dialog(axipy.Style.from_mapinfo('Symbol (35, 0, 12)'))
```

24.23 Виджеты Аксиомы

24.23.1 LayerControlWidget - Виджет управления слоями карты

class axipy.LayerControlWidget

Виджет управления слоями карты. Список доступных виджетов можно получить посредством `axipy.ViewManager.layer_controls`

Пример добавления в главное окно приложения (полный пример см. [Пример приложения из готовых компонентов](#)):

```
from PySide2.QtWidgets import QDockWidget
from PySide2.QtCore import Qt

layer_control = LayerControlWidget()
dock_layer_control = QDockWidget('Управление слоями', app.mainwindow.widget)
dock_layer_control.setWidget(layer_control.widget)
app.mainwindow.widget.addDockWidget(Qt.LeftDockWidgetArea, dock_layer_control)
```

Пример запроса выделенных слоев:

```
for lc in axipy.view_manager.layer_controls:
    print(f'Карта: {lc.active_map_view.title}')
    if lc.selected_layers:
        for l in lc.selected_layers:
            print('Слой или группа: ' , l.title)
```

См.также:

[Пример приложения из готовых компонентов](#)

Свойства:

<code>active_map_view</code>	Возвращает активную карту.
<code>selected_layers</code>	Возвращает список выбранных слоев или групп.
<code>widget</code>	Виджет, соответствующий содержимому окна.г.

Сигналы:

<code>mapview_activated</code>	Сигнал об изменении активной карты.
<code>selection_changed</code>	Сигнал об изменении выбранных слоев или групп.

property active_map_view: Optional[MapView]

Возвращает активную карту.

property mapview_activated: Signal

Сигнал об изменении активной карты.

Тип результата

Signal[MapView]

property selected_layers: Optional[ListLayers]

Возвращает список выбранных слоев или групп.

property selection_changed: Signal

Сигнал об изменении выбранных слоев или групп.

Пример:

```
for lc in axipy.view_manager.layer_controls:
    lc.selection_changed.connect(lambda: print('Выбор изменен'))
```

Тип результата

Signal[]

property widget: QWidget

Виджет, соответствующий содержимому окна.г.

Результат

Qt5 виджет содержимого.

24.23.2 DataManagerWidget - Виджет управления списком открытых данных

class axipy.DataManagerWidget

Виджет управления списком открытых данных. Список доступных виджетов можно получить посредством `axipy.ViewManager.data_managers`

Пример добавления в главное окно приложения (полный пример см. [Пример приложения из готовых компонентов](#)):

```
from PySide2.QtWidgets import QDockWidget
from PySide2.QtCore import Qt

data_manager = DataManagerWidget()
dock_catalog = QDockWidget("Открытые данные")
dock_catalog.setWidget(data_manager.widget)
app.mainwindow.widget.addDockWidget(Qt.LeftDockWidgetArea, dock_catalog)
```

См.также:

[Пример приложения из готовых компонентов](#)

Свойства:

<code>list_widget</code>	Виджет, соответствующий содержанию списка таблиц.
<code>objects</code>	Список выделенных объектов.
<code>widget</code>	Виджет, соответствующий содержанию окна.

Сигналы:

<code>selection_changed</code>	Выделение в списке было изменено.

property list_widget: `QListView`

Виджет, соответствующий содержанию списка таблиц.

Результат

Qt5 виджет списка.

property objects: `List[DataObject]`

Список выделенных объектов.

property selection_changed: `Signal`

Выделение в списке было изменено.

Тип результата

`Signal[]`

property widget: `QWidget`

Виджет, соответствующий содержанию окна.

Результат

Qt5 виджет содержимого.

24.23.3 ViewManagerWidget - Виджет списка окон

class ахіру.ViewManagerWidget

Виджет списка окон.

См.также:

[Пример приложения из готовых компонентов](#)

Свойства:

<code>widget</code>	Виджет, соответствующий содержимому окна.

Сигналы:

<code>view_changed</code>	Сигнал об изменении выделения в списке.

property view_changed: Signal

Сигнал об изменении выделения в списке.

Тип результата

Signal[View]

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.23.4 NotificationWidget - Виджет уведомлений

class axipy.NotificationWidget

Виджет уведомлений.

См.также:

[Пример приложения из готовых компонентов](#)

Свойства:

<code>widget</code>	Виджет, соответствующий содержимому окна.

property widget: QWidget

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.23.5 PythonConsoleWidget - Виджет ввода команд python

class axipy.PythonConsoleWidget

Виджет ввода команд python.

См.также:

[Пример приложения из готовых компонентов](#)

Свойства:

<code>widget</code>	Виджет, соответствующий содержимому окна.

property widget: `QWidget`

Виджет, соответствующий содержимому окна.

Результат

Qt5 виджет содержимого.

24.24 Notifications - Отправление уведомлений

class `axipy.Notifications`

Отправление уведомлений в виде всплывающего окна с его последующей регистрацией в окне уведомлений.

Классовые методы:

<code>push(title, text[, type_message])</code>	Отправляет уведомление.

Атрибуты:

<code>Critical</code>	Критическая ошибка.
<code>Information</code>	Информационное сообщение.
<code>Success</code>	Успешное выполнение процесса.
<code>Warning</code>	Предупреждение.

Critical

Критическая ошибка.

Information

Информационное сообщение.

Устанавливается по умолчанию.

Success

Успешное выполнение процесса.

Warning

Предупреждение.

static `push(title: str, text: str, type_message: Notifications = Information)`

Отправляет уведомление.

Параметры

- **title** - Заголовок
- **text** - Текст сообщения.
- **type_message** - Тип сообщения. В зависимости от типа сообщения в окне уведомлений оно помечается соответствующим цветом.

Пример:

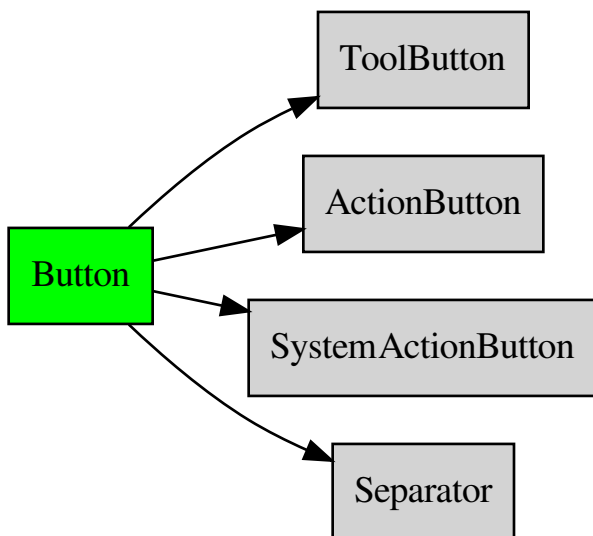
```
import axipy

axipy.Notifications.push('Предупреждение', 'Сообщение', axipy.Notifications.
↳Warning)
```

axipy.menubar - Модуль меню главного окна ГИС Аксиома.

Модуль меню главного окна ГИС Аксиома.

25.1 Button - Кнопка



class axipy.Button

Кнопка с инструментом для добавления в меню. Абстрактный класс.

Для создания объекта этого класса используйте [ActionButton](#), [ToolButton](#), [SystemActionButton](#).

Свойства:

<code>action</code>	Возвращает ссылку на объект <code>PySide2.QtWidgets.QAction</code> .
<code>observer</code>	Возвращает наблюдатель для определения доступности кнопки.
<code>observer_id</code>	Возвращает идентификатор наблюдателя для определения доступности кнопки.

Методы:

<code>remove()</code>	Удаляет кнопку из меню.

property action: `QAction`

Возвращает ссылку на объект `PySide2.QtWidgets.QAction`. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса `PySide2.QtWidgets.QAction`:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer: `Optional[Observer]`

Возвращает наблюдатель для определения доступности кнопки.

property observer_id: `str`

Возвращает идентификатор наблюдателя для определения доступности кнопки.

`remove()`

Удаляет кнопку из меню.

25.2 ActionButton - Кнопка с действием

`class` `axipy.ActionButton`

Базовые классы: `Button`

Кнопка с действием.

См.также:

`axipy.ObserverManager`.

Список 1: Пример со встроенным наблюдателем.

```
import axipy

button = axipy.ActionButton(
    title="Мое действие",
    on_click=lambda: print("clicked"),
    enable_on=axipy.ObserverManager.HasTables,
)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Добавление кнопки на панель "Основные", на вкладку "Команды"
position = axipy.Position("Основные", "Команды")
position.add(button)
# Чтобы удалить добавленную кнопку нужно вызвать remove, или перезапустить_
↪ приложение.
# button.remove()
```

Список 2: Пример со пользовательским наблюдателем.

```
import axipy

my_observer = axipy.Observer("MyObserver", False)
button = axipy.ActionButton(
    title="Мое действие",
    on_click=lambda: print("clicked"),
    enable_on=my_observer,
)

# Добавление кнопки на панель "Основные", на вкладку "Команды"
position = axipy.Position("Основные", "Команды")
position.add(button)
# Чтобы удалить добавленную кнопку нужно вызвать remove, или перезапустить_
↪ приложение.
# button.remove()
```

Конструктор класса:

<code>__init__(title, on_click[, icon, enable_on, ...])</code>	Конструктор класса.

Свойства:

<code>action</code>	Возвращает ссылку на объект <code>PySide2.QtWidgets.QAction</code> .
<code>observer</code>	Возвращает наблюдатель для определения доступности кнопки.
<code>observer_id</code>	Возвращает идентификатор наблюдателя для определения доступности кнопки.

Методы:

<code>remove()</code>	Удаляет кнопку из меню.

```
__init__(title: str, on_click: Callable[[], Any], icon: Union[str, Path, QIcon] = "",
         enable_on: Optional[Observer] = None, tooltip: Optional[str] = None,
         doc_file: Optional[Union[str, Path]] = None)
```

Конструктор класса.

Параметры

- **title** – Текст.

- **on_click** – Действие на нажатие. Передается функция, которая будет вызвана при нажатии на кнопку.
- **icon** – Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** – Наблюдатель, для определения доступности кнопки.
- **tooltip** – Строка с дополнительной короткой информацией по данному действию.
- **doc_file** – Полный путь к html файлу с документацией.

property action: QAction

Возвращает ссылку на объект `PySide2.QtWidgets.QAction`. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса `PySide2.QtWidgets.QAction`:

```
button.action.setTooltip("Всплывающая подсказка")
```

property observer: Optional[Observer]

Возвращает наблюдатель для определения доступности кнопки.

property observer_id: str

Возвращает идентификатор наблюдателя для определения доступности кнопки.

remove()

Удаляет кнопку из меню.

25.3 QActionButton - Действие, присутствующее в системе

class axipy.QActionButton

Базовые классы: `Button`

Кнопка для действия, присутствующего в системе.

См.также:

Примеры использования см. `axipy.ActionManager`

Конструктор класса:

<code>__init__(name)</code>	Конструктор класса.

Методы:

<code>remove()</code>	Удаляет кнопку из меню.

<code>__init__(name: str)</code>	Конструктор класса.

Параметры

name – идентификатор действия. Полный список доступных идентификаторов можно получить посредством `axipy.action_manager.keys()`.

remove()

Удаляет кнопку из меню.

25.4 ToolButton - Кнопка с инструментом

class `axipy.ToolButton`

Базовые классы: `Button`

Переключаемая кнопка с инструментом.

См.также:

`axipy.ViewTool`, `axipy.ObserverManager`.

Список 3: Пример

```
from typing import Optional

import axipy

# Класс инструмента
class MyTool(axipy.ViewTool):
    def __init__(self, param: str) -> None:
        self._param: str = param

def on_create_tool(view_type: axipy.ViewTool.ViewType) -> Optional[axipy.
↳ViewTool]:
    if view_type == axipy.ViewTool.ViewType.MAP_VIEW:
        # Если необходимо передавать параметры в инструмент, то их можно передать
↳в конструктор инструмента
        # внутри функции:
        param = "Передаваемый параметр"
        return MyTool(param)
    return None

# Создание кнопки с инструментом
button = axipy.ToolButton(
    title="Мой инструмент",
    on_create_tool=on_create_tool,
)

# Добавление кнопки на панель "Основные", на вкладку "Команды"
position = axipy.Position("Основные", "Команды")
position.add(button)
# Чтобы удалить добавленную кнопку нужно вызвать remove, или перезапустить
↳приложение.
# button.remove()
```

Конструктор класса:

`__init__(title, on_create_tool[, icon, ...])` Конструктор класса.

Свойства:

<code>action</code>	Возвращает ссылку на объект <code>PySide2.QtWidgets.QAction</code> .
<code>is_toggled</code>	Возвращает признак, нажата ли кнопка.
<code>observer</code>	Возвращает наблюдатель для определения доступности кнопки.
<code>observer_id</code>	Возвращает идентификатор наблюдателя для определения доступности кнопки.

Методы:

<code>remove()</code>	Удаляет кнопку из меню.

`__init__` (title: str, on_create_tool: Callable[[ViewType], Optional[ViewTool]], icon: Union[str, Path, QIcon] = "", enable_on: Optional[Observer] = None, tooltip: Optional[str] = None, doc_file: Optional[Union[str, Path]] = None, on_toggle: Optional[Callable[[bool], None]] = None)

Конструктор класса.

Параметры

- **title** - Текст.
- **on_create_tool** - Функция создающая инструмент. Передается функция, которая будет вызвана в момент создания инструмента для карты. Функция должна принимать обязательный параметр типа `axipy.ViewTool.ViewType` и возвращать экземпляр `axipy.ViewTool` или `None`.
- **icon** - Иконка. Может быть путем к файлу или адресом ресурса.
- **enable_on** - Наблюдатель, для определения доступности кнопки.
- **tooltip** - Строка с дополнительной короткой информацией по данному действию.
- **doc_file** - Полный путь к html файлу с документацией.
- **on_toggle** - Действие на нажатие. Передается функция, которая будет вызвана при нажатии на кнопку.

property action: QAction

Возвращает ссылку на объект `PySide2.QtWidgets.QAction`. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса `PySide2.QtWidgets.QAction`:

```
button.action.setToolTip("Всплывающая подсказка")
```

property is_toggled: bool

Возвращает признак, нажата ли кнопка.

property observer: `Optional[Observer]`

Возвращает наблюдатель для определения доступности кнопки.

property observer_id: `str`

Возвращает идентификатор наблюдателя для определения доступности кнопки.

remove()

Удаляет кнопку из меню.

25.5 Separator - Разделитель

class `axipy.Separator`

Базовые классы: `Button`

Разделитель.

Конструктор класса:

<code>__init__()</code>	Конструктор класса.

Свойства:

<code>action</code>	Возвращает ссылку на объект <code>PySide2.QtWidgets.QAction</code> .
<code>observer</code>	Возвращает наблюдатель для определения доступности кнопки.
<code>observer_id</code>	Возвращает идентификатор наблюдателя для определения доступности кнопки.

Методы:

<code>remove()</code>	Удаляет кнопку из меню.

`__init__()`

Конструктор класса.

property action: `QAction`

Возвращает ссылку на объект `PySide2.QtWidgets.QAction`. Через него можно производить дополнительные необходимые действия через объект Qt.

Пример задания всплывающей подсказки, используя метод класса `PySide2.QtWidgets.QAction`:

```
button.action.setToolTip("Всплывающая подсказка")
```

property observer: `Optional[Observer]`

Возвращает наблюдатель для определения доступности кнопки.

property observer_id: str

Возвращает идентификатор наблюдателя для определения доступности кнопки.

remove()

Удаляет кнопку из меню.

25.6 Position - Положение кнопки

class ахіру.Position

Положение кнопки в меню.

Пример:

```
import ахіру  
  
pos = ахіру.Position("Основные", "Команды")
```

Конструктор класса:

<code>__init__(tab, group)</code>	Конструктор класса.

Методы:

<code>add(button[, size])</code>	Добавляет кнопку в текущее положение.

`__init__(tab: str, group: str)`

Конструктор класса.

Параметры

- **tab** – Название вкладки.
- **group** – Название группы.

`add(button: Button, size: int = 2)`

Добавляет кнопку в текущее положение.

Параметры

- **button** – Кнопка.
- **size** – Размер кнопки. Маленькая кнопка - 1. Большая кнопка - 2.

Метод экземпляра

Функция, определенная в классе. Вызывается из экземпляра класса (объекта).

При определении функции, первый аргумент функции содержит ссылку на экземпляр класса и обычно называется `self`. При вызове функции, ссылка на экземпляр класса передается автоматически (неявно).

Классовый метод

Функция, определенная в классе и обернутая декоратором `classmethod()`. Вызывается из класса (типа).

При определении функции, первый аргумент функции содержит ссылку на класс (тип) и обычно называется `cls`. При вызове функции, ссылка на класс (тип) передается автоматически (неявно).

27.1 7.1 Изменения

Октябрь 2025

27.1.1 Новое

- Добавлен новый класс `axipy.SelectedFilter` — контейнер для передачи и получения фильтра, выбранного пользователем, в функциях: `axipy.open_file_dialog()`, `axipy.open_files_dialog()`, `axipy.save_file_dialog()` (параметр `selected_filter`).
- Реализован метод `axipy.SqliteDataProvider.get_destination()`.
- Добавлен новый класс `axipy.SqliteGeometryFormat` для указания формата геометрии SQLite в методе `axipy.SqliteDataProvider.get_destination()`.
- В `axipy.WmsDataProvider` добавлен параметр `color` (цвет подложки растра) в методы `axipy.WmsDataProvider.get_source()` и `axipy.WmsDataProvider.open()`.
- В `axipy.OgrDataProvider.get_destination()` в параметр `create_data` был добавлен ключ `driver` для явного указания имени провайдера.

27.2 7.0 Изменения

Июнь 2025

27.2.1 Новое

- В `axipy.open_file_dialog()` добавлен новый параметр `set_last_open_path`.
- В `axipy.open_files_dialog()` добавлен новый параметр `set_last_open_path`.
- В `axipy.save_file_dialog()` добавлен новый параметр `set_last_save_path`.
- Добавлена функция `axipy.select_folder_dialog()`.
- Добавлен метод `axipy.Rect.intersects()`.
- Добавлен метод `axipy.ProviderManager.find_for_extension()`.
- Для наследников класса `axipy.DatabaseDataProvider` добавлен параметр `editable_unique` (установить возможность изменения значения поля `unique`) в методы `axipy.DatabaseDataProvider.get_source()` и `axipy.DatabaseDataProvider.open()`.
- Добавлен новый класс `axipy.WebOpenData` (дополнительные данные открытия WEB-источников) для использования в методах `axipy.WmsDataProvider.open()`, `axipy.WmsDataProvider.get_source()`, `axipy.TmsDataProvider.open()`, `axipy.TmsDataProvider.get_source()`.
- В `axipy.ShapeDataProvider.get_destination()` добавлен новый параметр `create_data` для передачи параметров OGR.
- Добавлен параметр `open_data` в методы `axipy.DataProvider.open()` и `axipy.DataProvider.get_source()` в провайдеры `axipy.GdalDataProvider`, `axipy.OgrDataProvider` для передачи параметров открытия GDAL/OGR.
- Добавлен параметр `create_data` в методы `axipy.DataProvider.create_open()` и `axipy.DataProvider.get_destination()` в провайдеры `axipy.GdalDataProvider`, `axipy.OgrDataProvider` для передачи параметров создания GDAL/OGR.
- В `axipy.DataObject` добавлены свойства: `axipy.DataObject.file_name` (путь к файлу источника) и `axipy.DataObject.is_valid` (проверка на возможное закрытие объекта).
- Добавлен классовой метод `axipy.Style.for_geometry_type()` для получения стиля по умолчанию для типа геометрии.
- Добавлен новый класс `axipy.ViewTool` вместо устаревшего класса `axipy.MapTool`. Новый класс предоставляет возможность работы как с окном карты, так и с окном отчета для пользовательских инструментов.
- В `axipy.ActionManager` добавлены методы `axipy.ActionManager.activate_selection_tool()` и `axipy.ActionManager.find_by_text()`.
- В `axipy.Button` добавлено свойство `axipy.Button.observer`, возвращающее экземпляр наблюдателя кнопки.
- В `axipy.ToolButton` добавлено свойство `axipy.ToolButton.is_toggled` (нажата ли кнопка).

27.2.2 Исправления

- Свойство `axipy.MainWindow.catalog` объявлено устаревшим; используйте готовый экземпляр `axipy.data_manager`.
- Метод `axipy.ProviderManager.query()` объявлен устаревшим; используйте метод `axipy.DataManager.query()`.
- Класс `axipy.Туре` объявлен устаревшим и был переименован в `axipy.GeometryType`.

27.3 6.2 Изменения

Январь 2025

27.3.1 Новое

- В менеджер `axipy.view_manager` добавлены доступные в в системе `axipy.LayerControlWidget` и `axipy.DataManagerWidget` через свойства `axipy.ViewManager.layer_controls` и `axipy.ViewManager.data_managers` соответственно.
- В менеджер `axipy.selection_manager` добавлена возможность выбора всех объектов слоя через метод `axipy.SelectionManager.set_all()`

27.4 6.1 Изменения

Октябрь 2024

27.4.1 Новое

- В методы `axipy.open_file_dialog()` и `axipy.save_file_dialog()` добавлены дополнительные параметры.
- Для главного окна `MainWindow` добавлен список доступных `PySide2.QtWidgets.QDockWidget` как свойство `axipy.MainWindow.dock_widgets`
- В `axipy.DataManager` добавлен сигнал об изменении таблицы `axipy.DataManager.table_data_changed`
- Задание области врезки для карты `axipy.MapView.clip` и элемента отчета `axipy.MapReportItem.clip`
- Для пользовательских меток добавлено выравнивание `axipy.CustomLabelProperties.position_align`, стиль текста `axipy.CustomLabelProperties.text_style`, стиль указки `axipy.CustomLabelProperties.arrow_style`
- Стиль по умолчанию для пользовательских меток карты `axipy.CustomLabels.set_default_arrow_type()` и для выноски `axipy.CustomLabels.set_default_end_type()`

27.4.2 Исправления

- При сохранении `axipy.Table.commit()` выдача более полной информации в случае ошибки
- В методах `axipy.Workspace.load_file()` и `axipy.Workspace.load_string()` добавлена возможность пропуска источников, у которых возникли проблемы при открытии

27.5 6.0 Изменения

Июнь 2024

27.5.1 Новое

- Новые классы для работы с длительными задачами: `axipy.Task`, `axipy.DialogTask` вместо устаревших классов `axipy.AxipyTask`, `axipy.AxipyAnyCallableTask`, `axipy.AxipyProgressHandler`, `axipy.ProgressGuiFlags`, `axipy.ProgressSpecification`.
- Класс `axipy.TaskManager` был дополнен как словарь только для чтения (`typing.Mapping`). Добавлены сигналы `axipy.TaskManager.added` и `axipy.TaskManager.removed`. Методы `axipy.TaskManager.start_task()`, `axipy.TaskManager.run_and_get()`, `axipy.TaskManager.run_in_gui()`, `axipy.TaskManager.generate_dialog_for_task()` были объявлены устаревшими.
- Класс `axipy.PluginManager` - Менеджер плагинов.
- Функция `axipy.save_file_dialog()` - Диалог сохранения файла, по аналогии с `axipy.open_file_dialog()`.

27.5.2 Исправления

- Экземпляры классов менеджеров были приведены к одному виду и оформлены в разделе [Менеджеры](#). Классы:
 - `axipy.ObserverManager`;
 - `axipy.ActionManager`.были дополнены как экземпляры. (Раньше были исключительно статическими классами.)

27.6 5.2 Изменения

Февраль 2024

27.6.1 Исправления

- Функция `callback` для методов `axipy.Destination.export()` и `axipy.Destination.export_from_table()` теперь игнорирует все возвращаемые значения, отличные от `False`.

27.7 5.1 Изменения

Июль 2023

27.7.1 Исправления

- Методы `axipy.Workspace.load_file()` и `axipy.Workspace.save_file()` теперь работают независимо от наличия главного окна Аксиомы. Методы `axipy.MainWindow.load_workspace()` и `axipy.MainWindow.save_workspace()` отмечены как устаревшие.

27.8 5.0.1 Изменения

Июль 2023

27.8.1 Исправления

- Для классов (словарей `dict`):
 - `axipy.ObserverManager`
 - `axipy.CurrentSettings`
 - `axipy.DefaultSettings`
 - `axipy.LinearUnits`
 - `axipy.AreaUnits`
 - Исправлена ошибка с неправильной работой оператора принадлежности `in`;
 - Исправлена некорректная работа метода `get()`, где не учитывалось значение по умолчанию;
 - Исправлена ошибка, где при обращении по индексу `[]` к несуществующему элементу, не генерировалось исключение `KeyError`.

27.9 5.0 Изменения

Июнь 2023

27.9.1 Новое

- Раздел [Формирование пользовательского приложения](#).
- Классы виджетов Аксиомы:
 - `axipy.DataManagerWidget`
 - `axipy.ViewManagerWidget`
 - `axipy.LayerControlWidget`
 - `axipy.NotificationWidget`
 - `axipy.PythonConsoleWidget`
- Класс `axipy.ActionManager` - Менеджер системных действий и инструментов.
- Классы `axipy.LinearUnits`, `axipy.AreaUnits`.
- Классы `axipy.ObserverManager`, `axipy.Observer` вместо устаревших классов `axipy.StateManager` и `axipy.ValueObserver`.
- Классы `axipy.FloatCoord`, `axipy.AngleCoord` вместо устаревших классов `axipy.FloatFormatter` и `axipy.CoordFormatter`.
- Классы `axipy.CurrentSettings`, `axipy.DefaultSettings` вместо устаревшего класса `axipy.Settings`.
- Класс `axipy.StyleButton` вместо устаревшего класса `axipy.StyledButton`.
- Класс `axipy.Plugin` вместо устаревших классов `axipy.AxiomaPlugin` и `axipy.AxiomaInterface`.
- Метод `axipy.execfile()`.
- Метод `axipy.open_file_dialog()`.

27.9.2 Исправления

- Класс `axipy.app.Notifications` перенесен в `axipy.gui` как `axipy.gui.Notifications`.
- В соответствии с [PEP 8#package-and-module-names](#) переименованы модули:
 - Модуль `axipy.concurrent.AxipyProgressHandler` переименован в `axipy.concurrent.axipy_progress_handler`.
 - Модуль `axipy.concurrent.Task` переименован в `axipy.concurrent.task`.
 - Модуль `axipy.concurrent.TaskManager` переименован в `axipy.concurrent.task_manager_`.
 - Модуль `axipy.concurrent.TaskUtils` переименован в `axipy.concurrent.task_utils`.
 - Модуль `axipy.cs.CoordSystem` переименован в `axipy.cs.coord_system`.

- Модуль `axipy.cs.CoordTransformer` переименован в `axipy.cs.coord_transformer`.
- Модуль `axipy.da.attribute_schema` переименован в `axipy.da.schema`.
- Модуль `axipy.da.DataManagerWrapper` переименован в `axipy.da.data_manager_`.
- Модуль `axipy.da.DataObjectWrapper` переименован в `axipy.da.data_object`.
- Модуль `axipy.da.FeatureWrapper` переименован в `axipy.da.feature`.
- Модуль `axipy.da.Geometry` переименован в `axipy.da.geometry`.
- Модуль `axipy.da.Style` переименован в `axipy.da.style`.
- Модуль `axipy.da.TabFile` переименован в `axipy.da.tab_file`.
- Модуль `axipy.gui.ActiveToolPanel` переименован в `axipy.gui.active_tool_panel`.
- Модуль `axipy.gui.DialogWrapper` переименован в `axipy.gui.dialog`.
- Модуль `axipy.gui.Notifications` переименован в `axipy.gui.notifications`.
- Модуль `axipy.gui.SelectionManagerWrapper` переименован в `axipy.gui.selection_manager_`.
- Модуль `axipy.gui.ToolWrapper` переименован в `axipy.gui.map_tool`.
- Модуль `axipy.gui.view_manager_wrapper` переименован в `axipy.gui.view_manager_`.
- Модуль `axipy.gui.ViewWrapper` переименован в `axipy.gui.view`.
- Модуль `axipy.gui.WidgetWrapper` переименован в `axipy.gui.widgets`.
- Модуль `axipy.gui.Workspace` переименован в `axipy.gui.workspace`.
- Модуль `axipy.mi.MIGeometry` переименован в `axipy.mi.mi_geometry`.
- Модуль `axipy.render.map` переименован в `axipy.render.map_`.

27.10 4.4 Изменения

Февраль 2023

27.10.1 Исправления

- Методы `axipy.da.CollectionStyle.point()`, `axipy.da.CollectionStyle.line()`, `axipy.da.CollectionStyle.polygon()`, `axipy.da.CollectionStyle.text()` реализованы как свойства.

27.11 4.3 Изменения

Декабрь 2022

27.11.1 Новое

- Модули устанавливаются в папку `installed_modules`.
- Явное указание секции `[general]` в файле метаданных модуля `manifest.ini` необязательно.
- Модули с [зависимостями](#).
- Обход ошибки при импорте библиотеки `Matplotlib`.
- Раздел [Среда разработки](#).
- Методы `axipy.gui.SelectToolHelpers.select_by_mouse()` и `axipy.gui.SelectToolHelpers.select_by_rect()` для выделения геометрий как в инструменте Выбор.
- Доступ к объектам данных в `axipy.da.DataManager` по имени как в словаре [dict](#).
- Редактируемый атрибут `axipy.da.Table.schema`.
- [Свойства подписей](#).
- Список загруженных провайдеров `axipy.da.ProviderManager.providers()`.

27.11.2 Исправления

- При выполнении конвертации `axipy.da.MifMidDataProvider.convert_to_tab()` терялись пространственные данные.
- Из-за похожести с `axipy.gui.MapView.device_rect` свойство `axipy.gui.View.rect` переименовано в `axipy.gui.View.position`.
- `axipy.gui.Map.to_image()` не учитывал ограничивающий прямоугольник.
- Метод `axipy.da.Geometry.from_json()` переименован в `axipy.da.Geometry.from_geojson()`.
- Класс `axipy.utl.Printer` переименован в `axipy.utl.FloatFormatter`.
- Для `axipy.da.DataManager.sql_dialect` сменен тип данных с `str` на `enum`.
- Свойство `axipy.render.Label.placementPolicy` вынесено как `enum` [axipy.render.LabelOverlap](#).

27.12 4.0 Изменения

Июнь 2022

27.12.1 Новое

- Метод создания и показа главного окна `axipy.app.MainWindow.show()`.

27.12.2 Исправления

- Свойство `axipy.gui.ReportView.scale()` переименовано в `axipy.gui.ReportView.view_scale`.

27.13 3.7.0 Изменения

Март 2022

27.13.1 Новое

- Методы `axipy.gui.MapTool.load()/axipy.gui.MapTool.unload()` класса `axipy.gui.MapTool`; метод `axipy.gui.MapTool.deactivate()` отмечен как устаревший.
- Метод `axipy.gui.MapTool.canDeactivate()` переименован в `axipy.gui.MapTool.canUnload()`.
- Функция поиска перевода `axipy.tr()`.

27.13.2 Исправления

- Изменены пределы для свойств `axipy.render.RasterLayer.brightness` и `axipy.render.RasterLayer.contrast` на диапазон (-100...100).

27.14 3.5.0 Изменения

Август 2021

27.14.1 Новое

- Новые вспомогательные методы в `axipy.gui.MapTool`.
- Объектно-ориентированный стиль создания кнопок `axipy.menubar.Button`.
- Механизм слежения за значениями `axipy.da.state_manager`.
- Распространение модулей в [архивах](#).
- Объявление модулей с наследованием от `axipy.AxiomaPlugin`.
- Каталог данных содержит таблицу выборки `axipy.da.DataCatalog.selection`.
- Менеджер для запуска и управления пользовательскими задачами `axipy.concurrent.TaskManager`.
- Добавлена панель активного инструмента `axipy.gui.ActiveToolPanel` в которую можно поместить графический элемент упрощающий работу с пользовательским инструментом.

27.14.2 Исправления

- Класс `axipy.da.Collection` переименован в `axipy.da.GeometryCollection`.
- Методы `axipy.da.DataCatalog.tables()`, `axipy.da.DataCatalog.objects()`, `axipy.da.DataCatalog.count()` реализованы как свойства. Метод `axipy.da.Schema.attribute_names()` так-же переделан как свойство.
- Убраны класс `axipy.cs.UnitService` и его экземпляр `axipy.cs.unit`. Их функционал перенесен в базовый класс `axipy.cs.EarthUnit`, который переименован в `axipy.cs.Unit`. Переименованы методы `axipy.cs.LinearUnit.list_all()`, `axipy.cs.AreaUnit.list_all()`.
- Переименован класс `axipy.da.DataCatalog` в `axipy.da.DataManager`
- Переименован класс `axipy.gui.ViewService` в `axipy.gui.ViewManager`
- Переименован класс `axipy.gui.SelectionService` в `axipy.gui.SelectionManager`
- Переименован класс `axipy.da.DataProviders` в `axipy.da.ProviderManager`
- Экземпляр класса `axipy.render.Map` `axipy.render.Map.unit` перенесен в класс `axipy.gui.MapView`.

27.15 3.0.0 Изменения

Апрель 2021

27.15.1 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 ГИС Аксиома `axipy.Settings`.
- Провайдеры данных `axipy.da.DataProviders` со специализированными параметрами для открытия/создания и импорта/экспорта: `axipy.da.DataProviders.tab`, `axipy.da.DataProviders.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()`.
- Удаление кнопок `axipy.menubar.remove()` приводит к удалению групп и вкладок `axipy.menubar.Position`, если они стали пустыми.

27.15.2 Исправления

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

27.16 2.9.0 Изменения

Декабрь 2020

27.16.1 Новое

- Первоначальный релиз.

Важно! Нижеприведённые условия являются офертой на заключение безвозмездного Лицензионного договора на предоставление права использования Программы для ЭВМ для указанного в оферте неопределённого круга лиц, намеренных осуществлять использование Программы на личном компьютере не для целей выполнения своих обязанностей по трудовому договору, а также для лиц, использующих Программу: для оценки возможностей Программы, для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором, для образовательных и научных целей.

Прочитайте внимательно приведённые ниже условия Лицензионного договора, прежде чем устанавливать, копировать или иным образом использовать Программу для ЭВМ.

Начало использования Программы для ЭВМ, как оно определено условиями приведённого ниже Лицензионного договора, означает Ваше присоединение к приведённому ниже Лицензионному договору и безоговорочное согласие с его условиями в соответствии со ст.1286 ГК РФ.

В случае если Вы не согласны с условиями приведённого ниже Лицензионного договора, не устанавливайте и не используйте Программу или её компоненты.

28.1 БЕЗВОЗМЕЗДНЫЙ ЛИЦЕНЗИОННЫЙ ДОГОВОР

Настоящий Лицензионный договор («Договор») заключён между лицом, присоединившимся к настоящему Договору путём начала использования Программы («Пользователь»), и ООО «ЭСТИ», являющимся обладателем исключительного права на Программу («Правообладатель»).

28.2 ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ, ИСПОЛЬЗУЕМЫЕ В ДОГОВОРЕ

Программа – представленная в объективной форме совокупность данных и команд, предназначенных для функционирования ЭВМ и других компьютерных устройств в целях получения определённого результата, включая подготовительные материалы, полученные в ходе разработки Программы для ЭВМ, и порождаемые ею аудиовизуальные отображения, а именно Программа для ЭВМ ГИС «Аксиома» (зарегистрирована в Едином реестре российских программ для электронных вычислительных машин и баз данных под номером №2174, свидетельство о государственной регистрации Программы для ЭВМ №2016614626).

Правообладатель – обладатель исключительного права на Программу – ООО «ЭСТИ».

Пользователь – физическое лицо (-Вы), которое устанавливает и/или использует Программу от своего лица или правомерно владеет копией Программы. Если Программа была загружена или иным способом использована от имени юридического лица, то под термином Пользователь (-Вы) далее подразумевается юридическое лицо, для которого Программа была загружена или иным способом использована, и которое поручило отдельному физическому лицу принять данное соглашение от своего лица.

Официальный сайт Правообладателя – интернет-сайт, находящийся по адресу: <https://axioma-gis.ru>

Инсталляция (установка) – запись Программы в память ЭВМ, обеспечивающая возможность её хранения и использования по функциональному назначению.

Программный дистрибутив – комплект файлов, загруженный с Официального сайта Правообладателя и предназначенный для Инсталляции и использования Программы Пользователем. Загрузка Программного дистрибутива означает загрузку самой Программы.

Лицензия – неисключительное непередаваемое право использования Программы по прямому функциональному назначению.

28.3 СТАТЬЯ 1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1. Настоящий Договор устанавливает условия предоставления Правообладателем Пользователю прав использования Программы.

1.2. Настоящий Договор в соответствии со ст.1286 ГК РФ и ст.434 ГК РФ является договором присоединения и заключается в электронной форме путём начала использования Программы Пользователем, как такое начало определено в настоящем Договоре.

1.3. Началом использования Программы Пользователем является совершение Пользователем действий, направленных на воспроизведение Программы, в том числе загрузка Программного дистрибутива с Официального сайта Правообладателя и инсталляция Программы в память ЭВМ или на иной носитель.

1.4. Пользователь присоединяется к настоящему Договору с момента совершения любого из действий, перечисленных в пункте 1.3 настоящего Договора.

28.4 СТАТЬЯ 2. ПРЕДМЕТ ДОГОВОРА

2.1. Правообладатель безвозмездно, на условиях простой (неисключительной) лицензии, предоставляет Пользователю непередаваемое право использования Программы по прямому функциональному назначению путём воспроизведения на всей территории Российской Федерации. Для использования Программы Пользователю разрешается скопировать Программный дистрибутив и установить Программу.

2.2. Права на использование Программы предоставляются на весь срок действия исключительного права.

2.3. Пользователями Программы на условиях настоящего Договора могут быть следующие юридические лица:

2.3.1. Лица, использующие Программу исключительно в ознакомительных целях для оценки её функциональных возможностей.

2.3.2. Лица, использующие Программу исключительно для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором.

2.3.3. Бюджетные учреждения высшего и среднего образования.

2.3.4. Автономные учреждения высшего и среднего образования.

2.3.5. ФГБУ «Российская академия наук».

2.3.6. Научные организации, находящиеся под научно-методическим руководством президиума ФГБУ «Российская академия наук».

2.4. Пользователями Программы на условиях настоящего Договора могут быть следующие физические лица:

2.4.1. Лица, использующие Программу исключительно на личном компьютере не для целей выполнения своих обязанностей по трудовому договору.

2.4.2. Лица, использующие Программу исключительно в ознакомительных целях для оценки её функциональных возможностей.

2.4.3. Лица, использующие Программу исключительно для регистрации Лицензии на Программу в соответствии с ранее заключённым Лицензионным договором.

2.4.4. Лица, являющиеся учащимися бюджетных учреждений высшего и среднего образования.

2.4.5. Лица, являющиеся учащимися автономных учреждений высшего и среднего образования.

2.4.6. Сотрудники бюджетных учреждений высшего и среднего образования.

2.4.7. Сотрудники автономных учреждений высшего и среднего образования.

2.4.8. Сотрудники ФГБУ «Российская академия наук».

2.4.9. Сотрудники научных организаций, находящихся под научно-методическим руководством президиума ФГБУ «Российская академия наук».

28.5 СТАТЬЯ 3. ДОПОЛНИТЕЛЬНЫЕ УСЛОВИЯ

3.1. Правообладатель гарантирует, что он обладает всеми законными основаниями для предоставления Пользователю права использования Программы в пределах, установленных настоящим соглашением.

3.2. Права на использование Программы предоставляется Пользователю на условиях «как есть». Правообладатель отказывается от любых гарантий на Программу, прямо указанных или подразумеваемых, включая, но не ограничиваясь гарантиями пригодности для определённой Пользователем цели, а также не гарантирует, что Программа будет отвечать персональным требованиям и ожиданиям Пользователя, предъявляемым им Программе, аналогам, стандартам, не описанным в сопроводительной документации или на Официальном сайте Правообладателя. Правообладатель не гарантирует, что Программа полностью свободна от ошибок и будет работать с иной конфигурацией ЭВМ, чем указанные на Официальном сайте Правообладателя и/или будет совместима с иными программами, установленными на ЭВМ Пользователя.

3.3. Правообладатель не несёт ответственности за прямой и/или косвенный ущерб, причинённый Пользователю, а также не возмещает Пользователю убытки (включая упущенную выгоду), понесённые Пользователем, в том числе, в результате ненадлежащего качества каналов связи общего пользования, политику обмена трафиком между провайдерами, нормальное функционирование сети Интернет, её частей или за качество линий связи, не имеющих отношения к собственным ресурсам Правообладателя, и за их доступность для Пользователя.

3.4. Пользователь гарантирует, что конечный пользователь (физическое лицо) использует Программу не для целей получения прибыли в рамках коммерческой деятельности юридических лиц, а исключительно для образовательных или учебных целей, либо для получения дохода исключительно физическим лицом, использующим Программу на личном компьютере.

3.5. Пользователь гарантирует, что до присоединения к настоящему Договору он ознакомился с описанием Программы, требованиями к системно-аппаратной платформе, и иной информацией, относящейся к Программе, размещённой на Официальном сайте Правообладателя.

3.6. Пользователь не вправе продавать, отчуждать, передавать иным образом права на использование Программы, Программный дистрибутив третьим лицам.

3.7. Пользователь соглашается с тем, что Программа, документация, как и все другие объекты авторского права, а также системы, идеи и методы работы, другая информация, которая содержится в Программе, товарные знаки являются объектами интеллектуальной собственности Правообладателя. Данный Договор не даёт Пользователю никаких прав на использование объектов интеллектуальной собственности, включая товарные знаки и знаки обслуживания Правообладателя, за исключением прав, предоставляемых настоящим Лицензионным договором.

3.8. Пользователь не вправе модифицировать или изменять Программу никаким способом. Запрещается удалять или изменять наименование Программы, присутствующие в Программе, документации или иных материалах, распространяемых

с Программой, знаки охраны авторского права или иные указания на Правообладателя, уведомления об авторских правах на любой копии Программы.

3.9. Использование Программы в составе программных продуктов Пользователя возможно исключительно с согласия Правообладателя, выраженного в отдельном письменном соглашении. При этом указание названия Программы и Правообладателя является обязательным.

3.10. Использование несколькими физическими лицами экземпляра Программы, в том числе инсталлированной на серверной ЭВМ, возможно только с согласия Правообладателя, выраженного в отдельном письменном соглашении.

3.11. При разработке Программы использовались открытые лицензии. Авторские права на открытое программное обеспечение сохраняются за соответствующими правообладателями. Указание на правообладателей и лицензионные соглашения, по которым предоставляется открытое программное обеспечение, имеются в папке, входящей в состав Программного дистрибутива.

28.6 СТАТЬЯ 4. ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ

4.1. Настоящее Лицензионное соглашение регулируется в соответствии с законодательством Российской Федерации.

4.2. Досудебный порядок урегулирования споров является обязательным для Сторон. Претензия, направляемая в рамках досудебного порядка разрешения споров, должна быть мотивирована и содержать в себе описание конкретных достоверных обстоятельств неисполнения/нарушения существенных условий настоящего Договора. Претензия, оформленная без соблюдения условий, содержащихся в настоящем пункте, не признаётся Сторонами в качестве документа, направленного на соблюдение досудебного порядка разрешения спора. Сторона, получившая претензию, обязана направить мотивированный ответ на неё другой Стороне в срок не позднее 50 (Пятидесяти) дней с момента получения претензии. В случае недостижения согласия спор передаётся на рассмотрение в судебный орган в соответствии с установленной законом подведомственностью по месту нахождения Правообладателя.

4.3. Все изменения и дополнения к настоящему Договору признаются действительными только в случае составления их в письменной форме и подписания уполномоченными представителями каждой из Сторон, если возможность иного изменения условий настоящего Договора не предусмотрена самим Договором.

4.4. Настоящий Договор может изменяться Правообладателем в одностороннем порядке. Уведомление Пользователя о внесённых изменениях в условия настоящего Соглашения публикуется на Официальном сайте Правообладателя. Указанные изменения в условиях Договора вступают в силу с даты их публикации, если иное не оговорено в соответствующей публикации.

4.5. В случае нарушения Пользователем какого-либо из условий настоящего Договора, Правообладатель вправе отказаться от настоящего Договора в одностороннем порядке и заблокировать использование Программы.

4.6. Если какое-либо положение настоящего Лицензионного договора будет признано аннулированным, недействительным, не имеющим юридической силы или незаконным, то остальные положения настоящего Лицензионного договора сохраняют свою полную силу и действие.

4.7. Условия настоящего Договора являются превалирующими над условиями какого-либо иного лицензионного договора в отношении той же Программы между Пользователем и Правообладателем, если только в таком лицензионном договоре прямо не предусмотрено, что его положения имеют приоритет над настоящим Договором.

28.7 СТАТЬЯ 5. РЕКВИЗИТЫ ПРАВООБЛАДАТЕЛЯ

Правообладатель	ООО «ЭСТИ»		
ИНН	7704613038	КПП	770401001
Юридический адрес	119002, г. Москва, пер. Сивцев Вражек, д. 29/16, пом.2		
Адрес для почтовой корреспонденции	119002, г. Москва, пер. Сивцев Вражек, д. 29/16, пом.2		
Телефон, E-mail	+7 499 241-42-06; info@axioma-gis.ru		

Содержание модулей Python

а

axipy, 129
axipy.app, 147
axipy.concurrent, 167
axipy.cs, 153
axipy.da, 197
axipy.gui, 635
axipy.menubar, 693
axipy.raster, 534
axipy.utl, 183

Алфавитный указатель

Non-alphabetical

`__bool__()` (метод `axipy.AngleCoord`), 194
`__bool__()` (метод `axipy.FloatCoord`), 191
`__contains__()` (метод `axipy.DataManager`), 261
`__contains__()` (метод `axipy.Feature`), 303
`__contains__()` (метод `axipy.GeometryCollection`), 379
`__contains__()` (метод `axipy.ListHoles`), 375
`__contains__()` (метод `axipy.ListPoint`), 374
`__contains__()` (метод `axipy.Map`), 546
`__contains__()` (метод `axipy.MultiLineString`), 405
`__contains__()` (метод `axipy.MultiPoint`), 392
`__contains__()` (метод `axipy.MultiPolygon`), 418
`__contains__()` (метод `axipy.ObserverManager`), 538
`__contains__()` (метод `axipy.Schema`), 306
`__contains__()` (метод `axipy.ViewManager`), 677
`__copy__()` (метод `axipy.CoordSystem`), 155
`__copy__()` (метод `axipy.Pnt`), 184
`__copy__()` (метод `axipy.Rect`), 187
`__deepcopy__()` (метод `axipy.CoordSystem`), 156
`__deepcopy__()` (метод `axipy.Pnt`), 184
`__deepcopy__()` (метод `axipy.Rect`), 187
`__enter__()` (метод `axipy.CosmeticTable`), 294
`__enter__()` (метод `axipy.DataObject`), 266
`__enter__()` (метод `axipy.QueryTable`), 277
`__enter__()` (метод `axipy.Raster`), 532
`__enter__()` (метод `axipy.RasteredTable`), 542
`__enter__()` (метод `axipy.SelectionTable`), 286
`__enter__()` (метод `axipy.Table`), 269
`__eq__()` (метод `axipy.AngleCoord`), 194
`__eq__()` (метод `axipy.Arc`), 466
`__eq__()` (метод `axipy.AreaUnit`), 164
`__eq__()` (метод `axipy.Attribute`), 309
`__eq__()` (метод `axipy.BarThematicLayer`), 598
`__eq__()` (метод `axipy.CollectionStyle`), 529
`__eq__()` (метод `axipy.CoordSystem`), 156
`__eq__()` (метод `axipy.CosmeticLayer`), 566
`__eq__()` (метод `axipy.CosmeticTable`), 294
`__eq__()` (метод `axipy.DataObject`), 266
`__eq__()` (метод `axipy.DensityThematicLayer`), 611
`__eq__()` (метод `axipy.DrawableView`), 653
`__eq__()` (метод `axipy.Ellipse`), 454
`__eq__()` (метод `axipy.Feature`), 303
`__eq__()` (метод `axipy.FillStyle`), 517
`__eq__()` (метод `axipy.FloatCoord`), 191
`__eq__()` (метод `axipy.Geometry`), 316
`__eq__()` (метод `axipy.GeometryCollection`), 379
`__eq__()` (метод `axipy.GeometryReportItem`), 621
`__eq__()` (метод `axipy.IndividualThematicLayer`), 606
`__eq__()` (метод `axipy.Layer`), 554
`__eq__()` (метод `axipy.LegendReportItem`), 628
`__eq__()` (метод `axipy.LegendView`), 671
`__eq__()` (метод `axipy.Line`), 339
`__eq__()` (метод `axipy.LinearUnit`), 162
`__eq__()` (метод `axipy.LineString`), 351
`__eq__()` (метод `axipy.LineStyle`), 514
`__eq__()` (метод `axipy.Map`), 547
`__eq__()` (метод `axipy.MapReportItem`), 623
`__eq__()` (метод `axipy.MapView`), 658
`__eq__()` (метод `axipy.MultiLineString`), 405
`__eq__()` (метод `axipy.MultiPoint`), 392
`__eq__()` (метод `axipy.MultiPolygon`), 418
`__eq__()` (метод `axipy.Observer`), 539
`__eq__()` (метод `axipy.ObserverManager`), 538
`__eq__()` (метод `axipy.PieThematicLayer`), 592
`__eq__()` (метод `axipy.Pnt`), 184
`__eq__()` (метод `axipy.Point`), 327
`__eq__()` (метод `axipy.PointCompatStyle`), 500
`__eq__()` (метод `axipy.PointFontStyle`), 505
`__eq__()` (метод `axipy.PointPictureStyle`), 509
`__eq__()` (метод `axipy.PointStyle`), 497
`__eq__()` (метод `axipy.Polygon`), 364
`__eq__()` (метод `axipy.PolygonStyle`), 520
`__eq__()` (метод `axipy.QueryTable`), 277
`__eq__()` (метод `axipy.RangeThematicLayer`), 585
`__eq__()` (метод `axipy.Raster`), 532
`__eq__()` (метод `axipy.RasteredTable`), 542
`__eq__()` (метод `axipy.RasterLayer`), 558
`__eq__()` (метод `axipy.RasterReportItem`), 625
`__eq__()` (метод `axipy.Rect`), 187
`__eq__()` (метод `axipy.Rectangle`), 430
`__eq__()` (метод `axipy.ReportItem`), 620
`__eq__()` (метод `axipy.ReportView`), 665
`__eq__()` (метод `axipy.RoundRectangle`), 442
`__eq__()` (метод `axipy.ScaleBarReportItem`), 629
`__eq__()` (метод `axipy.SelectionTable`), 286
`__eq__()` (метод `axipy.Style`), 493
`__eq__()` (метод `axipy.SymbolThematicLayer`), 602
`__eq__()` (метод `axipy.Table`), 269
`__eq__()` (метод `axipy.TableReportItem`), 627
`__eq__()` (метод `axipy.TableView`), 650
`__eq__()` (метод `axipy.Text`), 479
`__eq__()` (метод `axipy.TextStyle`), 524

__eq__() (метод axipy.UnitValue), 165
 __eq__() (метод axipy.VectorLayer), 562
 __eq__() (метод axipy.View), 648
 __exit__() (метод axipy.CosmeticTable), 295
 __exit__() (метод axipy.DataObject), 266
 __exit__() (метод axipy.QueryTable), 278
 __exit__() (метод axipy.Raster), 532
 __exit__() (метод axipy.RasteredTable), 542
 __exit__() (метод axipy.SelectionTable), 287
 __exit__() (метод axipy.Table), 270
 __format__() (метод axipy.Pnt), 184
 __format__() (метод axipy.Rect), 187
 __ge__() (метод axipy.AngleCoord), 194
 __ge__() (метод axipy.FloatCoord), 191
 __ge__() (метод axipy.UnitValue), 165
 __getitem__() (метод axipy.ActionManager), 681
 __getitem__() (метод axipy.DataManager), 261
 __getitem__() (метод axipy.Feature), 303
 __getitem__() (метод axipy.GeometryCollection), 379
 __getitem__() (метод axipy.ListHoles), 375
 __getitem__() (метод axipy.ListLayers), 551
 __getitem__() (метод axipy.ListLegend), 673
 __getitem__() (метод axipy.ListLegendItems), 579
 __getitem__() (метод axipy.ListPoint), 374
 __getitem__() (метод axipy.ListThematic), 569
 __getitem__() (метод axipy.MultiLineString), 405
 __getitem__() (метод axipy.MultiPoint), 392
 __getitem__() (метод axipy.MultiPolygon), 418
 __getitem__() (метод axipy.ObserverManager), 538
 __getitem__() (метод axipy.ReportItems), 618
 __getitem__() (метод axipy.Schema), 307
 __getitem__() (метод axipy.TaskManager), 181
 __getitem__() (метод axipy.ViewManager), 677
 __gt__() (метод axipy.AngleCoord), 194
 __gt__() (метод axipy.FloatCoord), 191
 __gt__() (метод axipy.UnitValue), 165
 __init__() (метод axipy.ActionButton), 695
 __init__() (метод axipy.AngleCoord), 194
 __init__() (метод axipy.Arc), 466
 __init__() (метод axipy.Attribute), 309
 __init__() (метод axipy.BoundingRectDialog), 686
 __init__() (метод axipy.ChooseCoordSystemDialog), 684
 __init__() (метод axipy.CoordTransformer), 159
 __init__() (метод axipy.DialogTask), 176
 __init__() (метод axipy.Ellipse), 454
 __init__() (метод axipy.Feature), 303
 __init__() (метод axipy.FillStyle), 517
 __init__() (метод axipy.FloatCoord), 191
 __init__() (метод axipy.Line), 339
 __init__() (метод axipy.LineString), 352
 __init__() (метод axipy.LineStyle), 514
 __init__() (метод axipy.MapReportItem), 623
 __init__() (метод axipy.Observer), 539
 __init__() (метод axipy.Pnt), 184
 __init__() (метод axipy.Point), 327
 __init__() (метод axipy.PointCompatStyle), 500
 __init__() (метод axipy.PointFontStyle), 505
 __init__() (метод axipy.PointPictureStyle), 509
 __init__() (метод axipy.Polygon), 364
 __init__() (метод axipy.PolygonStyle), 520
 __init__() (метод axipy.Position), 700
 __init__() (метод axipy.Rect), 187
 __init__() (метод axipy.Rectangle), 430
 __init__() (метод axipy.RoundRectangle), 442
 __init__() (метод axipy.Schema), 307
 __init__() (метод axipy.Separator), 699
 __init__() (метод axipy.StyleButton), 687
 __init__() (метод axipy.SystemActionButton), 696
 __init__() (метод axipy.Task), 170
 __init__() (метод axipy.Text), 479
 __init__() (метод axipy.TextStyle), 524
 __init__() (метод axipy.ToolButton), 698
 __init__() (метод axipy.UnitValue), 165
 __iter__() (метод axipy.ActionManager), 681
 __iter__() (метод axipy.CosmeticTable), 295
 __iter__() (метод axipy.DataManager), 261
 __iter__() (метод axipy.Feature), 303
 __iter__() (метод axipy.GeometryCollection), 379
 __iter__() (метод axipy.ListHoles), 375
 __iter__() (метод axipy.ListLayers), 551
 __iter__() (метод axipy.ListLegend), 673
 __iter__() (метод axipy.ListLegendItems), 579
 __iter__() (метод axipy.ListPoint), 374
 __iter__() (метод axipy.ListThematic), 569
 __iter__() (метод axipy.MultiLineString), 405
 __iter__() (метод axipy.MultiPoint), 392
 __iter__() (метод axipy.MultiPolygon), 418
 __iter__() (метод axipy.ObserverManager), 538
 __iter__() (метод axipy.QueryTable), 278
 __iter__() (метод axipy.ReportItems), 618
 __iter__() (метод axipy.SelectionTable), 287
 __iter__() (метод axipy.Table), 270
 __iter__() (метод axipy.TaskManager), 181
 __iter__() (метод axipy.ViewManager), 677
 __le__() (метод axipy.AngleCoord), 195
 __le__() (метод axipy.FloatCoord), 192
 __le__() (метод axipy.UnitValue), 165
 __len__() (метод axipy.ActionManager), 681
 __len__() (метод axipy.CosmeticTable), 295
 __len__() (метод axipy.DataManager), 261
 __len__() (метод axipy.Feature), 303
 __len__() (метод axipy.GeometryCollection), 379
 __len__() (метод axipy.IndividualThematicLayer), 606
 __len__() (метод axipy.LineString), 352
 __len__() (метод axipy.ListHoles), 375
 __len__() (метод axipy.ListLayers), 551
 __len__() (метод axipy.ListLegend), 673
 __len__() (метод axipy.ListLegendItems), 579
 __len__() (метод axipy.ListPoint), 374
 __len__() (метод axipy.ListThematic), 569
 __len__() (метод axipy.MultiLineString), 405
 __len__() (метод axipy.MultiPoint), 392
 __len__() (метод axipy.MultiPolygon), 418
 __len__() (метод axipy.ObserverManager), 538
 __len__() (метод axipy.Polygon), 364
 __len__() (метод axipy.QueryTable), 278
 __len__() (метод axipy.ReportItems), 618
 __len__() (метод axipy.Schema), 307
 __len__() (метод axipy.SelectionManager), 674
 __len__() (метод axipy.SelectionTable), 287
 __len__() (метод axipy.Table), 270
 __len__() (метод axipy.TaskManager), 181
 __len__() (метод axipy.ViewManager), 677
 __lt__() (метод axipy.AngleCoord), 195
 __lt__() (метод axipy.FloatCoord), 192
 __lt__() (метод axipy.UnitValue), 166
 __repr__() (метод axipy.AngleCoord), 195
 __repr__() (метод axipy.Arc), 466
 __repr__() (метод axipy.AreaUnit), 164
 __repr__() (метод axipy.Attribute), 309
 __repr__() (метод axipy.BarThematicLayer), 598
 __repr__() (метод axipy.CollectionStyle), 529
 __repr__() (метод axipy.CoordSystem), 156
 __repr__() (метод axipy.CosmeticLayer), 566
 __repr__() (метод axipy.CosmeticTable), 295

__repr__() (метод axipy.CsvDataProvider), 212
__repr__() (метод axipy.CustomLabelProperties), 575
__repr__() (метод axipy.DatabaseDataProvider), 230
__repr__() (метод axipy.DataObject), 266
__repr__() (метод axipy.DensityThematicLayer), 611
__repr__() (метод axipy.DialogTask), 176
__repr__() (метод axipy.DrawableView), 653
__repr__() (метод axipy.DwgDataProvider), 253
__repr__() (метод axipy.Ellipse), 454
__repr__() (метод axipy.ExcelDataProvider), 214
__repr__() (метод axipy.FillStyle), 517
__repr__() (метод axipy.FloatCoord), 192
__repr__() (метод axipy.GdalDataProvider), 249
__repr__() (метод axipy.Geometry), 316
__repr__() (метод axipy.GeometryCollection), 380
__repr__() (метод axipy.IndividualThematicLayer), 606
__repr__() (метод axipy.Layer), 555
__repr__() (метод axipy.LegendView), 671
__repr__() (метод axipy.Line), 340
__repr__() (метод axipy.LinearUnit), 162
__repr__() (метод axipy.LineString), 352
__repr__() (метод axipy.LineStyle), 514
__repr__() (метод axipy.ListThematic), 569
__repr__() (метод axipy.MapView), 658
__repr__() (метод axipy.MifMidDataProvider), 216
__repr__() (метод axipy.MsSqlDataProvider), 236
__repr__() (метод axipy.MultiLineString), 405
__repr__() (метод axipy.MultiPoint), 393
__repr__() (метод axipy.MultiPolygon), 418
__repr__() (метод axipy.Observer), 539
__repr__() (метод axipy.ObserverManager), 538
__repr__() (метод axipy.OgrDataProvider), 251
__repr__() (метод axipy.OracleDataProvider), 233
__repr__() (метод axipy.PanoramaDataProvider), 257
__repr__() (метод axipy.PieThematicLayer), 592
__repr__() (метод axipy.Pnt), 184
__repr__() (метод axipy.Point), 328
__repr__() (метод axipy.PointCompatStyle), 501
__repr__() (метод axipy.PointFontStyle), 505
__repr__() (метод axipy.PointPictureStyle), 510
__repr__() (метод axipy.PointStyle), 497
__repr__() (метод axipy.Polygon), 364
__repr__() (метод axipy.PolygonStyle), 520
__repr__() (метод axipy.PostgreDataProvider), 227
__repr__() (метод axipy.QueryTable), 278
__repr__() (метод axipy.RangeThematicLayer), 585
__repr__() (метод axipy.Raster), 532
__repr__() (метод axipy.RasteredTable), 542
__repr__() (метод axipy.RasterLayer), 558
__repr__() (метод axipy.Rect), 187
__repr__() (метод axipy.Rectangle), 431
__repr__() (метод axipy.ReportView), 665
__repr__() (метод axipy.RestDataProvider), 242
__repr__() (метод axipy.RoundRectangle), 442
__repr__() (метод axipy.Schema), 307
__repr__() (метод axipy.SelectionTable), 287
__repr__() (метод axipy.ShapeDataProvider), 218
__repr__() (метод axipy.SqliteDataProvider), 220
__repr__() (метод axipy.Style), 493
__repr__() (метод axipy.SvgDataProvider), 225
__repr__() (метод axipy.SymbolThematicLayer), 602
__repr__() (метод axipy.TabDataProvider), 222
__repr__() (метод axipy.Table), 270
__repr__() (метод axipy.TableView), 650
__repr__() (метод axipy.Task), 170
__repr__() (метод axipy.Text), 479
__repr__() (метод axipy.TextStyle), 525
__repr__() (метод axipy.TmsDataProvider), 239
__repr__() (метод axipy.UnitValue), 166
__repr__() (метод axipy.VectorLayer), 562
__repr__() (метод axipy.View), 648
__repr__() (метод axipy.WmsDataProvider), 244
__repr__() (метод axipy.WmtsDataProvider), 247
setitem__() (метод axipy.Feature), 303
setitem__() (метод axipy.GeometryCollection), 380
setitem__() (метод axipy.ListLegendItems), 579
setitem__() (метод axipy.ListPoint), 375
setitem__() (метод axipy.MultiLineString), 405
setitem__() (метод axipy.MultiPoint), 393
setitem__() (метод axipy.MultiPolygon), 418
setitem__() (метод axipy.Schema), 307
str__() (метод axipy.AngleCoord), 195
str__() (метод axipy.AreaUnit), 164
str__() (метод axipy.BarThematicLayer), 598
str__() (метод axipy.CollectionStyle), 529
str__() (метод axipy.CoordSystem), 156
str__() (метод axipy.CosmeticLayer), 566
str__() (метод axipy.DensityThematicLayer), 612
str__() (метод axipy.DialogTask), 176
str__() (метод axipy.DrawableView), 653
str__() (метод axipy.FillStyle), 517
str__() (метод axipy.FloatCoord), 192
str__() (метод axipy.GeometryCollection), 380
str__() (метод axipy.IndividualThematicLayer), 606
str__() (метод axipy.Layer), 555
str__() (метод axipy.LegendView), 671
str__() (метод axipy.LinearUnit), 162
str__() (метод axipy.LineString), 352
str__() (метод axipy.LineStyle), 514
str__() (метод axipy.ListPoint), 375
str__() (метод axipy.Map), 547
str__() (метод axipy.MapView), 658
str__() (метод axipy.MultiLineString), 405
str__() (метод axipy.MultiPoint), 393
str__() (метод axipy.MultiPolygon), 418
str__() (метод axipy.PieThematicLayer), 592
str__() (метод axipy.Pnt), 184
str__() (метод axipy.Point), 328
str__() (метод axipy.PointCompatStyle), 501
str__() (метод axipy.PointFontStyle), 505
str__() (метод axipy.PointPictureStyle), 510
str__() (метод axipy.PointStyle), 497
str__() (метод axipy.Polygon), 364
str__() (метод axipy.PolygonStyle), 520
str__() (метод axipy.RangeThematicLayer), 585
str__() (метод axipy.RasterLayer), 558
str__() (метод axipy.Rect), 188
str__() (метод axipy.ReportView), 665
str__() (метод axipy.Style), 493
str__() (метод axipy.SymbolThematicLayer), 602
str__() (метод axipy.TableView), 650
str__() (метод axipy.Task), 170
str__() (метод axipy.TextStyle), 525
str__() (метод axipy.UnitValue), 166
str__() (метод axipy.VectorLayer), 562
str__() (метод axipy.View), 648

К

Классовый метод, 701

М

Метод экземпляра, 701

модуль

ахипу, 129

ахипу.app, 147

axipy.concurrent, 167
 axipy.cs, 153
 axipy.da, 197
 axipy.gui, 635
 axipy.menubar, 693
 axipy.raster, 534
 axipy.utl, 183

П

Предложения об улучшениях Python

PEP 8#Imports, 118
 PEP 8#package-and-module-names, 708

А

ACCEPT (атрибут axipy.ViewTool), 637
 accepted (axipy.AxipyAcceptableActiveToolHandler property), 646
 action (axipy.ActionButton property), 696
 action (axipy.Button property), 694
 action (axipy.Separator property), 699
 action (axipy.ToolButton property), 698
 action_manager (в модуле axipy), 142
 ActionButton (класс в axipy), 694
 ActionManager (класс в axipy), 680
 activate() (метод axipy.ActionManager), 681
 activate() (метод axipy.AxipyActiveToolPanelHandlerBase), 644
 activate() (метод axipy.ViewManager), 677
 activate_selection_tool() (метод axipy.ActionManager), 682
 activated (axipy.AxipyActiveToolPanelHandlerBase property), 644
 active (axipy.ViewManager property), 678
 active_changed (axipy.ViewManager property), 678
 active_map_view (axipy.LayerControlWidget property), 689
 ActiveMapView (атрибут axipy.ObserverManager), 537
 ActiveTableView (атрибут axipy.ObserverManager), 537
 ActiveToolPanel (класс в axipy), 642
 ActiveView (атрибут axipy.ObserverManager), 537
 actual_size (axipy.PointPictureStyle property), 510
 add() (метод axipy.DataManager), 261
 add() (метод axipy.MainWindow), 148
 add() (метод axipy.Position), 700
 add() (метод axipy.ReportItems), 618
 add() (метод axipy.SelectionManager), 674
 add_dock_widget() (метод axipy.MainWindow), 148
 add_group() (метод axipy.ListLayers), 551
 add_layer_current_map() (метод axipy.MainWindow), 149
 add_layer_interactive() (метод axipy.MainWindow), 149
 add_layer_new_map() (метод axipy.MainWindow), 149
 add_to_current_mapview() (метод axipy.ViewManager), 678
 added (axipy.DataManager property), 261
 added (axipy.TaskManager property), 181
 address (axipy.PasswordDialog property), 685
 affine_transform() (метод axipy.Arc), 466
 affine_transform() (метод axipy.Ellipse), 454
 affine_transform() (метод axipy.Geometry), 316
 affine_transform() (метод axipy.GeometryCollection), 380
 affine_transform() (метод axipy.Line), 340
 affine_transform() (метод axipy.LineString), 352

affine_transform() (метод axipy.MultiLineString), 405
 affine_transform() (метод axipy.MultiPoint), 393
 affine_transform() (метод axipy.MultiPolygon), 418
 affine_transform() (метод axipy.Point), 328
 affine_transform() (метод axipy.Polygon), 364
 affine_transform() (метод axipy.Rectangle), 431
 affine_transform() (метод axipy.RoundRectangle), 442
 affine_transform() (метод axipy.Text), 479
 Algorithm (класс в axipy), 534
 alias (axipy.Attribute property), 309
 alignment (axipy.TextStyle property), 525
 all_objects (axipy.DataManager property), 261
 AllocationThematic (класс в axipy), 613
 allocationType (axipy.AllocationThematic property), 614
 allocationType (axipy.BarThematicLayer property), 598
 allocationType (axipy.PieThematicLayer property), 592
 almost_equals() (метод axipy.Arc), 467
 almost_equals() (метод axipy.Ellipse), 455
 almost_equals() (метод axipy.Geometry), 317
 almost_equals() (метод axipy.GeometryCollection), 381
 almost_equals() (метод axipy.Line), 341
 almost_equals() (метод axipy.LineString), 353
 almost_equals() (метод axipy.MultiLineString), 406
 almost_equals() (метод axipy.MultiPoint), 394
 almost_equals() (метод axipy.MultiPolygon), 419
 almost_equals() (метод axipy.Point), 329
 almost_equals() (метод axipy.Polygon), 365
 almost_equals() (метод axipy.Rectangle), 432
 almost_equals() (метод axipy.RoundRectangle), 443
 almost_equals() (метод axipy.Text), 480
 angle (axipy.CustomLabelProperties property), 575
 angle (axipy.Text property), 480
 AngleCoord (класс в axipy), 193
 append() (метод axipy.GeometryCollection), 381
 append() (метод axipy.ListHoles), 376
 append() (метод axipy.ListLayers), 551
 append() (метод axipy.ListPoint), 375
 append() (метод axipy.ListThematic), 569
 append() (метод axipy.MultiLineString), 407
 append() (метод axipy.MultiPoint), 394
 append() (метод axipy.MultiPolygon), 420
 apply_color (axipy.PointPictureStyle property), 510
 Arc (класс в axipy), 463
 areaInterior (axipy.Label property), 571
 areaLayout (axipy.Label property), 571
 areaPosition (axipy.Label property), 571
 AreaUnit (класс в axipy), 163
 areaUnit (axipy.Map property), 547
 AreaUnits (класс в axipy), 162
 arrange() (метод axipy.LegendView), 671
 arrow_style (axipy.CustomLabelProperties property), 575
 as_float_round() (метод axipy.FloatCoord), 192
 as_float_round_signific() (метод axipy.FloatCoord), 192
 as_rumb() (метод axipy.AngleCoord), 195
 as_string() (метод axipy.AngleCoord), 195
 as_string() (метод axipy.FloatCoord), 192
 assign_gray() (метод axipy.IndividualThematicLayer), 606
 assign_gray() (метод axipy.RangeThematicLayer), 585

- assign_gray() (метод axipy.ReallocateThematicColor), 581
 - assign_monotone() (метод axipy.IndividualThematicLayer), 606
 - assign_monotone() (метод axipy.RangeThematicLayer), 585
 - assign_monotone() (метод axipy.ReallocateThematicColor), 581
 - assign_rainbow() (метод axipy.IndividualThematicLayer), 607
 - assign_rainbow() (метод axipy.RangeThematicLayer), 585
 - assign_rainbow() (метод axipy.ReallocateThematicColor), 581
 - assign_three_colors() (метод axipy.IndividualThematicLayer), 607
 - assign_three_colors() (метод axipy.RangeThematicLayer), 585
 - assign_three_colors() (метод axipy.ReallocateThematicColor), 581
 - assign_two_colors() (метод axipy.IndividualThematicLayer), 607
 - assign_two_colors() (метод axipy.RangeThematicLayer), 586
 - assign_two_colors() (метод axipy.ReallocateThematicColor), 582
 - at() (метод axipy.ListLayers), 551
 - at() (метод axipy.ListThematic), 569
 - at() (метод axipy.ReportItems), 618
 - Attribute (класс в axipy), 308
 - attribute_names (axipy.Schema property), 307
 - AxiomaInitLogLevel (класс в axipy), 129
 - AxiomaLanguage (класс в axipy), 139
 - axipy
 - модуль, 129
 - AxipyAcceptableActiveToolHandler (класс в axipy), 645
 - AxipyActiveToolPanelHandlerBase (класс в axipy), 643
 - axipy.app
 - модуль, 147
 - axipy.concurrent
 - модуль, 167
 - axipy.cs
 - модуль, 153
 - AxipyCustomActiveToolPanelHandler (класс в axipy), 646
 - axipy.da
 - модуль, 197
 - axipy.gui
 - модуль, 635
 - axipy.menubar
 - модуль, 693
 - axipy.raster
 - модуль, 534
 - axipy.utl
 - модуль, 183
- B**
- backgroundColor (axipy.Label property), 571
 - backgroundSize (axipy.Label property), 571
 - backgroundType (axipy.Label property), 571
 - BarThematicLayer (класс в axipy), 595
 - base_table (axipy.SelectionTable property), 287
 - begin (axipy.Line property), 341
 - bg_color (axipy.FillStyle property), 517
 - bg_color (axipy.TextStyle property), 525
 - bg_type (axipy.TextStyle property), 525
 - black_border (axipy.PointFontStyle property), 505
 - bold (axipy.PointFontStyle property), 506
 - bold (axipy.TextStyle property), 525
 - bool() (статический метод axipy.Attribute), 310
 - border (axipy.PolygonStyle property), 520
 - border_style (axipy.GeometryReportItem property), 621
 - border_style (axipy.Legend property), 577
 - border_style (axipy.LegendReportItem property), 628
 - border_style (axipy.MapReportItem property), 623
 - border_style (axipy.RasterReportItem property), 625
 - border_style (axipy.ReportItem property), 620
 - border_style (axipy.ScaleBarReportItem property), 629
 - border_style (axipy.TableReportItem property), 627
 - boundary() (метод axipy.Arc), 468
 - boundary() (метод axipy.Ellipse), 456
 - boundary() (метод axipy.Geometry), 317
 - boundary() (метод axipy.GeometryCollection), 381
 - boundary() (метод axipy.Line), 341
 - boundary() (метод axipy.LineString), 353
 - boundary() (метод axipy.MultiLineString), 407
 - boundary() (метод axipy.MultiPoint), 394
 - boundary() (метод axipy.MultiPolygon), 420
 - boundary() (метод axipy.Point), 329
 - boundary() (метод axipy.Polygon), 366
 - boundary() (метод axipy.Rectangle), 432
 - boundary() (метод axipy.RoundRectangle), 444
 - boundary() (метод axipy.Text), 480
 - BoundingRectDialog (класс в axipy), 685
 - bounds (axipy.Arc property), 468
 - bounds (axipy.Ellipse property), 456
 - bounds (axipy.Geometry property), 317
 - bounds (axipy.GeometryCollection property), 381
 - bounds (axipy.Line property), 341
 - bounds (axipy.LineString property), 353
 - bounds (axipy.MultiLineString property), 407
 - bounds (axipy.MultiPoint property), 394
 - bounds (axipy.MultiPolygon property), 420
 - bounds (axipy.Point property), 329
 - bounds (axipy.Polygon property), 366
 - bounds (axipy.Rectangle property), 432
 - bounds (axipy.RoundRectangle property), 444
 - bounds (axipy.Text property), 480
 - brightness (axipy.RasterLayer property), 558
 - BrushCatalog (атрибут axipy.CurrentSettings), 136
 - buffer (axipy.ClipGeometry property), 632
 - buffer() (метод axipy.Arc), 468
 - buffer() (метод axipy.Ellipse), 456
 - buffer() (метод axipy.Geometry), 317
 - buffer() (метод axipy.GeometryCollection), 381
 - buffer() (метод axipy.Line), 341
 - buffer() (метод axipy.LineString), 353
 - buffer() (метод axipy.MultiLineString), 407
 - buffer() (метод axipy.MultiPoint), 394
 - buffer() (метод axipy.MultiPolygon), 420
 - buffer() (метод axipy.Point), 329
 - buffer() (метод axipy.Polygon), 366
 - buffer() (метод axipy.Rectangle), 432
 - buffer() (метод axipy.RoundRectangle), 444
 - buffer() (метод axipy.Text), 480
 - Button (класс в axipy), 693
 - by_name() (метод axipy.Schema), 307
- C**
- calc_mode (axipy.MapView property), 658
 - CalcMode (класс в axipy), 196

callout (axipy.TextStyle property), 525
callout_style (axipy.TextStyle property), 525
can_redo (axipy.CosmeticTable property), 295
can_redo (axipy.DrawableView property), 653
can_redo (axipy.MapView property), 659
can_redo (axipy.QueryTable property), 278
can_redo (axipy.ReportView property), 665
can_redo (axipy.SelectionTable property), 287
can_redo (axipy.Table property), 270
can_undo (axipy.CosmeticTable property), 295
can_undo (axipy.DrawableView property), 653
can_undo (axipy.MapView property), 659
can_undo (axipy.QueryTable property), 278
can_undo (axipy.ReportView property), 665
can_undo (axipy.SelectionTable property), 287
can_undo (axipy.Table property), 270
can_unload() (метод axipy.ViewTool), 637
cancel() (метод axipy.DialogTask), 176
cancel() (метод axipy.Task), 170
cancelable (axipy.DialogTask property), 176
capital (axipy.TextStyle property), 525
caption (axipy.Legend property), 577
CENTER (атрибут axipy.BarThematicLayer), 597
CENTER (атрибут axipy.OrientationThematic), 614
CENTER (атрибут axipy.PieThematicLayer), 591
center (axipy.Arc property), 468
center (axipy.Ellipse property), 456
center (axipy.MapReportItem property), 623
center (axipy.MapView property), 659
center (axipy.Rect property), 188
centroid() (метод axipy.Arc), 468
centroid() (метод axipy.Ellipse), 456
centroid() (метод axipy.Geometry), 318
centroid() (метод axipy.GeometryCollection), 382
centroid() (метод axipy.Line), 341
centroid() (метод axipy.LineString), 354
centroid() (метод axipy.MultiLineString), 407
centroid() (метод axipy.MultiPoint), 395
centroid() (метод axipy.MultiPolygon), 420
centroid() (метод axipy.Point), 329
centroid() (метод axipy.Polygon), 366
centroid() (метод axipy.Rectangle), 432
centroid() (метод axipy.RoundRectangle), 444
centroid() (метод axipy.Text), 481
change_coordsystem() (метод axipy.TabDataProvider), 222
changed (axipy.Observer property), 539
changed (axipy.SelectionManager property), 674
check_query() (метод axipy.DataManager), 261
ChooseCoordSystemDialog (класс в axipy), 684
chosenCoordSystem() (метод axipy.ChooseCoordSystemDialog), 685
class_geometries() (метод axipy.CosmeticTable), 295
class_geometries() (метод axipy.QueryTable), 278
class_geometries() (метод axipy.SelectionTable), 287
class_geometries() (метод axipy.Table), 270
clear() (метод axipy.SelectionManager), 674
clear_guidelines() (метод axipy.ReportView), 665
clear_selected_guidelines() (метод axipy.ReportView), 666
clip (axipy.Context property), 630
clip (axipy.MapReportItem property), 623
clip (axipy.MapView property), 659
ClipGeometry (класс в axipy), 632
clone() (метод axipy.Arc), 468
clone() (метод axipy.CollectionStyle), 529
clone() (метод axipy.CoordSystem), 156
clone() (метод axipy.Ellipse), 456
clone() (метод axipy.FillStyle), 517
clone() (метод axipy.Geometry), 318
clone() (метод axipy.GeometryCollection), 382
clone() (метод axipy.Line), 341
clone() (метод axipy.LineString), 354
clone() (метод axipy.LineStyle), 514
clone() (метод axipy.MultiLineString), 407
clone() (метод axipy.MultiPoint), 395
clone() (метод axipy.MultiPolygon), 420
clone() (метод axipy.Pnt), 184
clone() (метод axipy.Point), 330
clone() (метод axipy.PointCompatStyle), 501
clone() (метод axipy.PointFontStyle), 506
clone() (метод axipy.PointPictureStyle), 510
clone() (метод axipy.PointStyle), 497
clone() (метод axipy.Polygon), 366
clone() (метод axipy.PolygonStyle), 521
clone() (метод axipy.Rect), 188
clone() (метод axipy.Rectangle), 432
clone() (метод axipy.RoundRectangle), 444
clone() (метод axipy.Style), 493
clone() (метод axipy.Text), 481
clone() (метод axipy.TextStyle), 525
close() (метод axipy.CosmeticTable), 295
close() (метод axipy.DataObject), 266
close() (метод axipy.DrawableView), 653
close() (метод axipy.LegendView), 671
close() (метод axipy.MapView), 659
close() (метод axipy.QueryTable), 278
close() (метод axipy.Raster), 533
close() (метод axipy.RasteredTable), 542
close() (метод axipy.ReportView), 666
close() (метод axipy.SelectionTable), 287
close() (метод axipy.Table), 270
close() (метод axipy.TableView), 650
close() (метод axipy.View), 648
close() (метод axipy.ViewManager), 678
close_all() (метод axipy.ViewManager), 678
CollectionStyle (класс в axipy), 528
color (axipy.DensityThematicLayer property), 612
color (axipy.FillStyle property), 517
color (axipy.Label property), 572
color (axipy.LineStyle property), 514
color (axipy.PointCompatStyle property), 501
color (axipy.PointFontStyle property), 506
color (axipy.PointPictureStyle property), 510
color (axipy.PointStyle property), 497
color (axipy.TextStyle property), 525
columns (axipy.Legend property), 578
columns (axipy.TableReportItem property), 627
comments (axipy.Attribute property), 310
commit() (метод axipy.CosmeticTable), 295
commit() (метод axipy.QueryTable), 278
commit() (метод axipy.SelectionTable), 287
commit() (метод axipy.Table), 270
compare() (статический метод axipy.Version), 150
Compression (класс в axipy), 535
contains() (метод axipy.Arc), 468
contains() (метод axipy.Ellipse), 456
contains() (метод axipy.Geometry), 318
contains() (метод axipy.GeometryCollection), 382
contains() (метод axipy.Line), 342
contains() (метод axipy.LineString), 354
contains() (метод axipy.MultiLineString), 407
contains() (метод axipy.MultiPoint), 395
contains() (метод axipy.MultiPolygon), 420
contains() (метод axipy.Point), 330
contains() (метод axipy.Polygon), 366

contains() (метод axipy.Rect), 188
contains() (метод axipy.Rectangle), 432
contains() (метод axipy.RoundRectangle), 444
contains() (метод axipy.Text), 481
Context (класс в ахипу), 630
contrast (axipy.RasterLayer property), 558
conversion (axipy.AreaUnit property), 164
conversion (axipy.LinearUnit property), 162
convert_file() (метод axipy.DwgDataProvider), 253
convert_file() (метод axipy.PanoramaDataProvider), 257
convert_from_degree() (метод axipy.CoordSystem), 156
convert_to_degree() (метод axipy.CoordSystem), 156
convert_to_tab() (метод axipy.MifMidDataProvider), 216
convex_hull() (метод axipy.Arc), 468
convex_hull() (метод axipy.Ellipse), 456
convex_hull() (метод axipy.Geometry), 318
convex_hull() (метод axipy.GeometryCollection), 382
convex_hull() (метод axipy.Line), 342
convex_hull() (метод axipy.LineString), 354
convex_hull() (метод axipy.MultiLineString), 408
convex_hull() (метод axipy.MultiPoint), 395
convex_hull() (метод axipy.MultiPolygon), 420
convex_hull() (метод axipy.Point), 330
convex_hull() (метод axipy.Polygon), 366
convex_hull() (метод axipy.Rectangle), 432
convex_hull() (метод axipy.RoundRectangle), 444
convex_hull() (метод axipy.Text), 481
CoordSystem (класс в ахипу), 153
coordsystem (axipy.Arc property), 468
coordsystem (axipy.BarThematicLayer property), 598
coordsystem (axipy.Context property), 631
coordsystem (axipy.CosmeticLayer property), 566
coordsystem (axipy.CosmeticTable property), 295
coordsystem (axipy.DensityThematicLayer property), 612
coordsystem (axipy.Ellipse property), 456
coordsystem (axipy.Geometry property), 318
coordsystem (axipy.GeometryCollection property), 382
coordsystem (axipy.IndividualThematicLayer property), 607
coordsystem (axipy.Layer property), 555
coordsystem (axipy.Line property), 342
coordsystem (axipy.LineString property), 354
coordsystem (axipy.MapReportItem property), 623
coordsystem (axipy.MapView property), 659
coordsystem (axipy.MultiLineString property), 408
coordsystem (axipy.MultiPoint property), 395
coordsystem (axipy.MultiPolygon property), 420
coordsystem (axipy.PieThematicLayer property), 592
coordsystem (axipy.Point property), 330
coordsystem (axipy.Polygon property), 366
coordsystem (axipy.QueryTable property), 278
coordsystem (axipy.RangeThematicLayer property), 586
coordsystem (axipy.Raster property), 533
coordsystem (axipy.RasteredTable property), 542
coordsystem (axipy.RasterLayer property), 558
coordsystem (axipy.Rectangle property), 433
coordsystem (axipy.RoundRectangle property), 444
coordsystem (axipy.Schema property), 307
coordsystem (axipy.SelectionTable property), 287
coordsystem (axipy.SymbolThematicLayer property), 602
coordsystem (axipy.Table property), 270
coordsystem (axipy.Text property), 481
coordsystem (axipy.VectorLayer property), 562
coordsystem_changed (axipy.MapView property), 659
coordsystem_visual (axipy.MapView property), 659
CoordTransformer (класс в ахипу), 159
copy_table_files() (метод axipy.TabDataProvider), 223
cosmetic (axipy.Map property), 547
CosmeticLayer (класс в ахипу), 564
CosmeticTable (класс в ахипу), 292
count (axipy.DataManager property), 262
count (axipy.IndividualThematicLayer property), 607
count (axipy.ListLayers property), 551
count (axipy.ListThematic property), 569
count (axipy.ReportItems property), 619
count (axipy.SelectionManager property), 674
count (axipy.ViewManager property), 678
count() (метод axipy.CosmeticTable), 295
count() (метод axipy.QueryTable), 278
count() (метод axipy.SelectionTable), 287
count() (метод axipy.Table), 270
count_changed (axipy.ViewManager property), 678
covers() (метод axipy.Arc), 468
covers() (метод axipy.Ellipse), 456
covers() (метод axipy.Geometry), 318
covers() (метод axipy.GeometryCollection), 382
covers() (метод axipy.Line), 342
covers() (метод axipy.LineString), 354
covers() (метод axipy.MultiLineString), 408
covers() (метод axipy.MultiPoint), 395
covers() (метод axipy.MultiPolygon), 421
covers() (метод axipy.Point), 330
covers() (метод axipy.Polygon), 366
covers() (метод axipy.Rectangle), 433
covers() (метод axipy.RoundRectangle), 444
covers() (метод axipy.Text), 481
create() (метод класса axipy.BarThematicLayer), 598
create() (метод класса axipy.CosmeticLayer), 566
create() (метод класса axipy.DensityThematicLayer), 612
create() (метод класса axipy.IndividualThematicLayer), 607
create() (метод класса axipy.Layer), 555
create() (метод класса axipy.PieThematicLayer), 592
create() (метод класса axipy.RangeThematicLayer), 586
create() (метод класса axipy.RasterLayer), 558
create() (метод класса axipy.SymbolThematicLayer), 602
create() (метод класса axipy.VectorLayer), 562
create() (метод axipy.ProviderManager), 198
create_action() (метод axipy.Plugin), 130
create_by_style() (метод класса axipy.Text), 481
create_legendview() (метод axipy.ViewManager), 678
create_mapview() (метод axipy.ViewManager), 678
create_mi_compat() (статический метод axipy.PointCompatStyle), 501
create_mi_compat() (статический метод axipy.PointFontStyle), 506
create_mi_compat() (статический метод axipy.PointPictureStyle), 510
create_mi_compat() (статический метод axipy.PointStyle), 497
create_mi_font() (статический метод axipy.PointCompatStyle), 501
create_mi_font() (статический метод axipy.PointFontStyle), 506
create_mi_font() (статический метод axipy.PointPictureStyle), 510

create_mi_font() (статический метод axipy.PointStyle), 497
 create_mi_picture() (статический метод axipy.PointCompatStyle), 501
 create_mi_picture() (статический метод axipy.PointFontStyle), 506
 create_mi_picture() (статический метод axipy.PointPictureStyle), 511
 create_mi_picture() (статический метод axipy.PointStyle), 497
 create_open() (метод axipy.CsvDataProvider), 212
 create_open() (метод axipy.DatabaseDataProvider), 230
 create_open() (метод axipy.DataProvider), 207
 create_open() (метод axipy.Destination), 209
 create_open() (метод axipy.DwgDataProvider), 254
 create_open() (метод axipy.ExcelDataProvider), 214
 create_open() (метод axipy.GdalDataProvider), 249
 create_open() (метод axipy.MifMidDataProvider), 217
 create_open() (метод axipy.MsSqlDataProvider), 236
 create_open() (метод axipy.OgrDataProvider), 251
 create_open() (метод axipy.OracleDataProvider), 233
 create_open() (метод axipy.PanoramaDataProvider), 257
 create_open() (метод axipy.PostgreDataProvider), 227
 create_open() (метод axipy.ProviderManager), 198
 create_open() (метод axipy.RestDataProvider), 242
 create_open() (метод axipy.ShapeDataProvider), 218
 create_open() (метод axipy.SqliteDataProvider), 220
 create_open() (метод axipy.SvgDataProvider), 225
 create_open() (метод axipy.TabDataProvider), 223
 create_open() (метод axipy.TmsDataProvider), 239
 create_open() (метод axipy.WmsDataProvider), 244
 create_open() (метод axipy.WmtsDataProvider), 247
 create_reportview() (метод axipy.ViewManager), 679
 create_tableview() (метод axipy.ViewManager), 679
 create_tool() (метод axipy.Plugin), 130
 create_view() (метод axipy.ViewManager), 679
 createfile() (метод axipy.ProviderManager), 199
 createIndex (атрибут axipy.ExportParameters), 211
 CreateTabAfterOpen (атрибут axipy.CurrentSettings), 136
 Critical (атрибут axipy.Notifications), 692
 crosses() (метод axipy.Arc), 468
 crosses() (метод axipy.Ellipse), 456
 crosses() (метод axipy.Geometry), 318
 crosses() (метод axipy.GeometryCollection), 382
 crosses() (метод axipy.Line), 342
 crosses() (метод axipy.LineString), 354
 crosses() (метод axipy.MultiLineString), 408
 crosses() (метод axipy.MultiPoint), 395
 crosses() (метод axipy.MultiPolygon), 421
 crosses() (метод axipy.Point), 330
 crosses() (метод axipy.Polygon), 366
 crosses() (метод axipy.Rectangle), 433
 crosses() (метод axipy.RoundRectangle), 444
 crosses() (метод axipy.Text), 482
 csv (axipy.ProviderManager property), 199
 CsvDataProvider (класс в axipy), 212
 current() (метод класса axipy.CoordSystem), 156
 CurrentSettings (класс в axipy), 134
 cursor (axipy.ViewTool property), 637
 custom_labels (axipy.Мap property), 547
 CustomLabelEndType (класс в axipy), 576
 CustomLabelProperties (класс в axipy), 575
 CustomLabels (класс в axipy), 631

D

data_changed (axipy.BarThematicLayer property), 598
 data_changed (axipy.CosmeticLayer property), 566
 data_changed (axipy.CosmeticTable property), 296
 data_changed (axipy.DensityThematicLayer property), 612
 data_changed (axipy.IndividualThematicLayer property), 608
 data_changed (axipy.Layer property), 555
 data_changed (axipy.PieThematicLayer property), 592
 data_changed (axipy.QueryTable property), 279
 data_changed (axipy.RangeThematicLayer property), 586
 data_changed (axipy.RasterLayer property), 558
 data_changed (axipy.SelectionTable property), 288
 data_changed (axipy.SymbolThematicLayer property), 602
 data_changed (axipy.Table property), 271
 data_changed (axipy.VectorLayer property), 562
 data_manager (в модуле axipy), 142
 data_managers (axipy.ViewManager property), 679
 data_object (axipy.BarThematicLayer property), 598
 data_object (axipy.CosmeticLayer property), 566
 data_object (axipy.DensityThematicLayer property), 612
 data_object (axipy.IndividualThematicLayer property), 608
 data_object (axipy.Layer property), 555
 data_object (axipy.PieThematicLayer property), 592
 data_object (axipy.RangeThematicLayer property), 586
 data_object (axipy.RasterLayer property), 558
 data_object (axipy.SymbolThematicLayer property), 602
 data_object (axipy.tableView property), 650
 data_object (axipy.VectorLayer property), 562
 DatabaseDataProvider (класс в axipy), 229
 DataManager (класс в axipy), 259
 DataManagerWidget (класс в axipy), 689
 DataObject (класс в axipy), 265
 DataProvider (класс в axipy), 206
 date() (статический метод axipy.Attribute), 310
 datetime() (статический метод axipy.Attribute), 310
 deactivate() (метод axipy.AxipyActiveToolPanelHandlerBase), 644
 deactivated (axipy.AxipyActiveToolPanelHandlerBase property), 644
 DeactivationReason (класс в axipy), 641
 decimal() (статический метод axipy.Attribute), 310
 default_calc_mode (axipy.CoordSystem property), 156
 DEFAULT_DECIMAL_LENGTH (атрибут axipy.Attribute), 309
 DEFAULT_DECIMAL_PRECISION (атрибут axipy.Attribute), 309
 DEFAULT_PORT (атрибут axipy.MsSqlDataProvider), 236
 DEFAULT_PORT (атрибут axipy.OracleDataProvider), 233
 DEFAULT_PORT (атрибут axipy.PostgreDataProvider), 227
 DEFAULT_STRING_LENGTH (атрибут axipy.Attribute), 309
 DefaultPathCache (атрибут axipy.CurrentSettings), 136
 DefaultSettings (класс в axipy), 139
 defaultStyle (axipy.SymbolThematicLayer property), 602
 degrees (axipy.AngleCoord property), 195
 delayed (axipy.DialogTask property), 176
 delete() (метод axipy.Schema), 307
 DensityThematicLayer (класс в axipy), 610
 description (axipy.AreaUnit property), 164

- ul style="list-style-type: none; padding-left: 0;">
- description (axipy.DialogTask property), 176
- description (axipy.LinearUnit property), 162
- description (axipy.Task property), 170
- Destination (класс в ахipy), 208
- destroyed (axipy.CosmeticTable property), 296
- destroyed (axipy.DataObject property), 267
- destroyed (axipy.QueryTable property), 279
- destroyed (axipy.Raster property), 533
- destroyed (axipy.RasteredTable property), 542
- destroyed (axipy.SelectionTable property), 288
- destroyed (axipy.Table property), 271
- device_rect (axipy.MapView property), 660
- device_to_scene_transform (axipy.DrawableView property), 653
- device_to_scene_transform (axipy.MapView property), 660
- device_to_scene_transform (axipy.Raster property), 533
- device_to_scene_transform (axipy.ReportView property), 666
- DialogTask (класс в ахipy), 173
- DialogTask.CanceledException, 175
- DialogTask.Range (класс в ахipy), 175
- DialogTask.Status (класс в ахipy), 176
- difference() (метод axipy.Arc), 468
- difference() (метод axipy.Ellipse), 456
- difference() (метод axipy.Geometry), 318
- difference() (метод axipy.GeometryCollection), 382
- difference() (метод axipy.Line), 342
- difference() (метод axipy.LineString), 354
- difference() (метод axipy.MultiLineString), 408
- difference() (метод axipy.MultiPoint), 395
- difference() (метод axipy.MultiPolygon), 421
- difference() (метод axipy.Point), 330
- difference() (метод axipy.Polygon), 366
- difference() (метод axipy.Rectangle), 433
- difference() (метод axipy.RoundRectangle), 444
- difference() (метод axipy.Text), 482
- disable() (метод axipy.AxipyAcceptableActiveToolHandler), 646
- disjoint() (метод axipy.Arc), 468
- disjoint() (метод axipy.Ellipse), 456
- disjoint() (метод axipy.Geometry), 318
- disjoint() (метод axipy.GeometryCollection), 382
- disjoint() (метод axipy.Line), 342
- disjoint() (метод axipy.LineString), 354
- disjoint() (метод axipy.MultiLineString), 408
- disjoint() (метод axipy.MultiPoint), 395
- disjoint() (метод axipy.MultiPolygon), 421
- disjoint() (метод axipy.Point), 330
- disjoint() (метод axipy.Polygon), 367
- disjoint() (метод axipy.Rectangle), 433
- disjoint() (метод axipy.RoundRectangle), 444
- disjoint() (метод axipy.Text), 482
- distance_by_points() (статический метод axipy.Arc), 468
- distance_by_points() (статический метод axipy.Ellipse), 457
- distance_by_points() (статический метод axipy.Geometry), 318
- distance_by_points() (статический метод axipy.GeometryCollection), 382
- distance_by_points() (статический метод axipy.Line), 342
- distance_by_points() (статический метод axipy.LineString), 354
- distance_by_points() (статический метод axipy.MultiLineString), 408
- distance_by_points() (статический метод axipy.MultiPoint), 395
- distance_by_points() (статический метод axipy.MultiPolygon), 421
- distance_by_points() (статический метод axipy.Point), 330
- distance_by_points() (статический метод axipy.Polygon), 367
- distance_by_points() (статический метод axipy.Rectangle), 433
- distance_by_points() (статический метод axipy.RoundRectangle), 445
- distance_by_points() (статический метод axipy.Text), 482
- DistancePrecision (атрибут axipy.CurrentSettings), 136
- distanceUnit (axipy.Map property), 547
- dock_widget_by_name() (метод axipy.MainWindow), 149
- dock_widgets (axipy.MainWindow property), 149
- DockWidgetArea (класс в ахipy), 150
- double() (статический метод axipy.Attribute), 310
- DOWN (атрибут axipy.BarThematicLayer), 597
- DOWN (атрибут axipy.OrientationThematic), 614
- DOWN (атрибут axipy.PieThematicLayer), 591
- dpi (axipy.Context property), 631
- draw() (метод axipy.CollectionStyle), 529
- draw() (метод axipy.FillStyle), 517
- draw() (метод axipy.Legend), 578
- draw() (метод axipy.LineStyle), 514
- draw() (метод axipy.Map), 547
- draw() (метод axipy.PointCompatStyle), 502
- draw() (метод axipy.PointFontStyle), 507
- draw() (метод axipy.PointPictureStyle), 511
- draw() (метод axipy.PointStyle), 498
- draw() (метод axipy.PolygonStyle), 521
- draw() (метод axipy.Report), 617
- draw() (метод axipy.Style), 493
- draw() (метод axipy.TextStyle), 525
- draw_vector() (метод axipy.Map), 548
- DrawableView (класс в ахipy), 652
- DrawCoordSysBounds (атрибут axipy.CurrentSettings), 137
- dropTable (атрибут axipy.ExportParameters), 211
- dwg (axipy.ProviderManager property), 199
- DwgDataProvider (класс в ахipy), 253
- DwgFileFormat (класс в ахipy), 255
- DwgFileVersion (класс в ахipy), 255
- DwgPalette (класс в ахipy), 256
- ## E
- Editable (атрибут axipy.ObserverManager), 537
 - editable_layer (axipy.Map property), 548
 - editable_layer (axipy.MapView property), 660
 - editable_layer_changed (axipy.MapView property), 660
 - EditNodeColor (атрибут axipy.CurrentSettings), 137
 - EditNodeSize (атрибут axipy.CurrentSettings), 137
 - Ellipse (класс в ахipy), 452
 - EnableSmartTabs (атрибут axipy.CurrentSettings), 137
 - end (axipy.Line property), 342
 - endAngle (axipy.Arc property), 469
 - endPoint (axipy.Text property), 483
 - endType (axipy.CustomLabelProperties property), 576
 - envelope() (метод axipy.Arc), 469
 - envelope() (метод axipy.Ellipse), 457
 - envelope() (метод axipy.Geometry), 319

envelope() (метод axipy.GeometryCollection), 383
 envelope() (метод axipy.Line), 342
 envelope() (метод axipy.LineString), 355
 envelope() (метод axipy.MultiLineString), 408
 envelope() (метод axipy.MultiPoint), 396
 envelope() (метод axipy.MultiPolygon), 421
 envelope() (метод axipy.Point), 330
 envelope() (метод axipy.Polygon), 367
 envelope() (метод axipy.Rectangle), 433
 envelope() (метод axipy.RoundRectangle), 445
 envelope() (метод axipy.Text), 483
 epsg (axipy.CoordSystem property), 156
 eq_approx() (метод класса axipy.Pnt), 184
 eq_approx() (метод класса axipy.Rect), 188
 EQUAL_COUNT (атрибут axipy.RangeThematicLayer), 584
 EQUAL_INTERVAL (атрибут axipy.RangeThematicLayer), 584
 equals() (метод axipy.Arc), 469
 equals() (метод axipy.Ellipse), 457
 equals() (метод axipy.Geometry), 319
 equals() (метод axipy.GeometryCollection), 383
 equals() (метод axipy.Line), 342
 equals() (метод axipy.LineString), 355
 equals() (метод axipy.MultiLineString), 408
 equals() (метод axipy.MultiPoint), 396
 equals() (метод axipy.MultiPolygon), 421
 equals() (метод axipy.Point), 330
 equals() (метод axipy.Polygon), 367
 equals() (метод axipy.Rectangle), 433
 equals() (метод axipy.RoundRectangle), 445
 equals() (метод axipy.Text), 483
 errorFile (атрибут axipy.ExportParameters), 211
 excel (axipy.ProviderManager property), 199
 ExcelDataProvider (класс в axipy), 214
 execfile() (в модуле axipy), 142
 exists() (метод axipy.DataManager), 262
 expanded() (метод axipy.Rect), 188
 export() (метод axipy.Destination), 209
 export_from() (метод axipy.Destination), 210
 export_from_table() (метод axipy.Destination), 210
 ExportParameters (класс в axipy), 211
 expression (axipy.CustomLabelProperties property), 576

F

Feature (класс в axipy), 301
 file_extensions() (метод axipy.CsvDataProvider), 213
 file_extensions() (метод axipy.DatabaseDataProvider), 230
 file_extensions() (метод axipy.DataProvider), 207
 file_extensions() (метод axipy.DwgDataProvider), 254
 file_extensions() (метод axipy.ExcelDataProvider), 214
 file_extensions() (метод axipy.GdalDataProvider), 249
 file_extensions() (метод axipy.MifMidDataProvider), 217
 file_extensions() (метод axipy.MsSqlDataProvider), 236
 file_extensions() (метод axipy.OgrDataProvider), 251
 file_extensions() (метод axipy.OracleDataProvider), 233
 file_extensions() (метод axipy.PanoramaDataProvider), 257
 file_extensions() (метод axipy.PostgreDataProvider), 227

file_extensions() (метод axipy.RestDataProvider), 242
 file_extensions() (метод axipy.ShapeDataProvider), 218
 file_extensions() (метод axipy.SqliteDataProvider), 220
 file_extensions() (метод axipy.SvgDataProvider), 225
 file_extensions() (метод axipy.TabDataProvider), 223
 file_extensions() (метод axipy.TmsDataProvider), 239
 file_extensions() (метод axipy.WmsDataProvider), 244
 file_extensions() (метод axipy.WmtsDataProvider), 247
 file_name (axipy.CosmeticTable property), 296
 file_name (axipy.DataObject property), 267
 file_name (axipy.QueryTable property), 279
 file_name (axipy.Raster property), 533
 file_name (axipy.RasteredTable property), 542
 file_name (axipy.SelectionTable property), 288
 file_name (axipy.Table property), 271
 filename (axipy.PointPictureStyle property), 511
 fill (axipy.PolygonStyle property), 521
 fill_on_pages() (метод axipy.Report), 617
 fill_on_pages() (метод axipy.ReportView), 666
 fill_style (axipy.GeometryReportItem property), 621
 fill_style (axipy.Legend property), 578
 fill_style (axipy.LegendReportItem property), 628
 fill_style (axipy.MapReportItem property), 624
 fill_style (axipy.RasterReportItem property), 625
 fill_style (axipy.ReportItem property), 620
 fill_style (axipy.ScaleBarReportItem property), 629
 fill_style (axipy.TableReportItem property), 627
 FillStyle (класс в axipy), 516
 filter (атрибут axipy.SelectedFilter), 140
 find() (метод axipy.DataManager), 262
 find_by_text() (метод axipy.ActionManager), 682
 find_for_extension() (метод axipy.ProviderManager), 199
 find_style() (метод axipy.CollectionStyle), 530
 finished (axipy.DialogTask property), 177
 finished (axipy.Task property), 170
 fit_pages() (метод axipy.Report), 617
 fixGeometry (атрибут axipy.ExportParameters), 211
 float() (статический метод axipy.Attribute), 310
 FloatCoord (класс в axipy), 191
 font (axipy.Label property), 572
 font_name (axipy.PointFontStyle property), 507
 fontname (axipy.TextStyle property), 526
 for_geometry() (метод класса axipy.CollectionStyle), 530
 for_geometry() (метод класса axipy.FillStyle), 518
 for_geometry() (метод класса axipy.LineStyle), 515
 for_geometry() (метод класса axipy.PointCompatStyle), 502
 for_geometry() (метод класса axipy.PointFontStyle), 507
 for_geometry() (метод класса axipy.PointPictureStyle), 511
 for_geometry() (метод класса axipy.PointStyle), 498
 for_geometry() (метод класса axipy.PolygonStyle), 521
 for_geometry() (метод класса axipy.Style), 494
 for_geometry() (метод класса axipy.TextStyle), 526
 for_geometry_type() (метод класса axipy.CollectionStyle), 530
 for_geometry_type() (метод класса axipy.FillStyle), 518

for_geometry_type() (метод класса axipy.LineStyle), 515
 for_geometry_type() (метод класса axipy.PointCompatStyle), 502
 for_geometry_type() (метод класса axipy.PointFontStyle), 507
 for_geometry_type() (метод класса axipy.PointPictureStyle), 511
 for_geometry_type() (метод класса axipy.PointStyle), 498
 for_geometry_type() (метод класса axipy.PolygonStyle), 521
 for_geometry_type() (метод класса axipy.Style), 494
 for_geometry_type() (метод класса axipy.TextStyle), 526
 for_line() (метод axipy.CollectionStyle), 530
 for_point() (метод axipy.CollectionStyle), 530
 for_polygon() (метод axipy.CollectionStyle), 530
 for_text() (метод axipy.CollectionStyle), 530
 Format (класс в axipy), 534
 from_area_unit() (статический метод axipy.LinearUnit), 162
 from_epsg() (метод класса axipy.CoordSystem), 156
 from_geojson() (статический метод axipy.Arc), 469
 from_geojson() (статический метод axipy.Ellipse), 457
 from_geojson() (статический метод axipy.Geometry), 319
 from_geojson() (статический метод axipy.GeometryCollection), 383
 from_geojson() (статический метод axipy.Line), 343
 from_geojson() (статический метод axipy.LineString), 355
 from_geojson() (статический метод axipy.MultiLineString), 408
 from_geojson() (статический метод axipy.MultiPoint), 396
 from_geojson() (статический метод axipy.MultiPolygon), 421
 from_geojson() (статический метод axipy.Point), 331
 from_geojson() (статический метод axipy.Polygon), 367
 from_geojson() (статический метод axipy.Rectangle), 433
 from_geojson() (статический метод axipy.RoundRectangle), 445
 from_geojson() (статический метод axipy.Text), 483
 from_linear_unit() (статический метод axipy.AreaUnit), 164
 from_mapinfo() (метод класса axipy.CollectionStyle), 530
 from_mapinfo() (метод класса axipy.FillStyle), 518
 from_mapinfo() (метод класса axipy.LineStyle), 515
 from_mapinfo() (метод класса axipy.PointCompatStyle), 502
 from_mapinfo() (метод класса axipy.PointFontStyle), 507
 from_mapinfo() (метод класса axipy.PointPictureStyle), 512
 from_mapinfo() (метод класса axipy.PointStyle), 498
 from_mapinfo() (метод класса axipy.PolygonStyle), 521
 from_mapinfo() (метод класса axipy.Style), 494
 from_mapinfo() (метод класса axipy.TextStyle), 526
 from_mif() (статический метод axipy.Arc), 469
 from_mif() (статический метод axipy.Ellipse), 457
 from_mif() (статический метод axipy.Geometry), 319
 from_mif() (статический метод axipy.GeometryCollection), 383
 from_mif() (статический метод axipy.Line), 343
 from_mif() (статический метод axipy.LineString), 355
 from_mif() (статический метод axipy.MultiLineString), 409
 from_mif() (статический метод axipy.MultiPoint), 396
 from_mif() (статический метод axipy.MultiPolygon), 421
 from_mif() (статический метод axipy.Point), 331
 from_mif() (статический метод axipy.Polygon), 367
 from_mif() (статический метод axipy.Rectangle), 434
 from_mif() (статический метод axipy.RoundRectangle), 445
 from_mif() (статический метод axipy.Text), 483
 from_parts() (метод класса axipy.AngleCoord), 195
 from_prj() (метод класса axipy.CoordSystem), 157
 from_proj() (метод класса axipy.CoordSystem), 157
 from_qt() (метод класса axipy.Pnt), 185
 from_qt() (метод класса axipy.Rect), 189
 from_rect() (статический метод axipy.Polygon), 367
 from_string() (метод класса axipy.CoordSystem), 157
 from_tuple() (метод класса axipy.Pnt), 185
 from_units() (метод класса axipy.CoordSystem), 157
 from_wkb() (метод класса axipy.Arc), 469
 from_wkb() (метод класса axipy.Ellipse), 457
 from_wkb() (метод класса axipy.Geometry), 319
 from_wkb() (метод класса axipy.GeometryCollection), 383
 from_wkb() (метод класса axipy.Line), 343
 from_wkb() (метод класса axipy.LineString), 355
 from_wkb() (метод класса axipy.MultiLineString), 409
 from_wkb() (метод класса axipy.MultiPoint), 396
 from_wkb() (метод класса axipy.MultiPolygon), 422
 from_wkb() (метод класса axipy.Point), 331
 from_wkb() (метод класса axipy.Polygon), 368
 from_wkb() (метод класса axipy.Rectangle), 434
 from_wkb() (метод класса axipy.RoundRectangle), 445
 from_wkb() (метод класса axipy.Text), 483
 from_wkt() (метод класса axipy.Arc), 470
 from_wkt() (метод класса axipy.CoordSystem), 157
 from_wkt() (метод класса axipy.Ellipse), 458
 from_wkt() (метод класса axipy.Geometry), 319
 from_wkt() (метод класса axipy.GeometryCollection), 383
 from_wkt() (метод класса axipy.Line), 343
 from_wkt() (метод класса axipy.LineString), 355
 from_wkt() (метод класса axipy.MultiLineString), 409
 from_wkt() (метод класса axipy.MultiPoint), 396
 from_wkt() (метод класса axipy.MultiPolygon), 422
 from_wkt() (метод класса axipy.Point), 331
 from_wkt() (метод класса axipy.Polygon), 368
 from_wkt() (метод класса axipy.Rectangle), 434
 from_wkt() (метод класса axipy.RoundRectangle), 446
 from_wkt() (метод класса axipy.Text), 484
 full_string() (статический метод axipy.Version), 151

G

GCP (класс в axipy), 534
 gdal (axipy.ProviderManager property), 199
 GdalDataProvider (класс в axipy), 248
 generate_dialog_for_task() (метод axipy.TaskManager), 181
 generate_tab() (метод axipy.TabFile), 540
 Geometry (класс в axipy), 313
 geometry (axipy.ClipGeometry property), 632
 geometry (axipy.Feature property), 303
 geometry (axipy.GeometryReportItem property), 621
 geometry (axipy.MainWindow property), 149
 geometryAsText (атрибут axipy.ExportParameters), 211

GeometryClass (клас в ахіру), 543
GeometryCollection (клас в ахіру), 376
geometryColumnName (атрибут ахіру.ExportParameters), 211
GeometryReportItem (клас в ахіру), 620
GeometryType (клас в ахіру), 490
get() (метод класа ахіру.CurrentSettings), 139
get() (метод ахіру.ActionManager), 682
get() (метод ахіру.CustomLabels), 631
get() (метод ахіру.Feature), 304
get() (метод ахіру.ObserverManager), 538
get() (метод ахіру.TaskManager), 181
get_area() (метод ахіру.Arc), 470
get_area() (метод ахіру.Ellipse), 458
get_area() (метод ахіру.Geometry), 320
get_area() (метод ахіру.GeometryCollection), 384
get_area() (метод ахіру.Line), 344
get_area() (метод ахіру.LineString), 356
get_area() (метод ахіру.MultiLineString), 409
get_area() (метод ахіру.MultiPoint), 397
get_area() (метод ахіру.MultiPolygon), 422
get_area() (метод ахіру.Point), 332
get_area() (метод ахіру.Polygon), 368
get_area() (метод ахіру.Rectangle), 434
get_area() (метод ахіру.RoundRectangle), 446
get_area() (метод ахіру.Text), 484
get_as_cursor() (метод ахіру.SelectionManager), 674
get_as_table() (метод ахіру.SelectionManager), 675
get_best_coordssystem() (метод ахіру.Map), 548
get_best_rect() (метод ахіру.Map), 548
get_bounds() (метод ахіру.BarThematicLayer), 598
get_bounds() (метод ахіру.CosmeticLayer), 566
get_bounds() (метод ахіру.CosmeticTable), 296
get_bounds() (метод ахіру.DensityThematicLayer), 612
get_bounds() (метод ахіру.IndividualThematicLayer), 608
get_bounds() (метод ахіру.Layer), 555
get_bounds() (метод ахіру.PieThematicLayer), 592
get_bounds() (метод ахіру.QueryTable), 279
get_bounds() (метод ахіру.RangeThematicLayer), 586
get_bounds() (метод ахіру.RasterLayer), 558
get_bounds() (метод ахіру.SelectionTable), 288
get_bounds() (метод ахіру.SymbolThematicLayer), 602
get_bounds() (метод ахіру.Table), 271
get_bounds() (метод ахіру.VectorLayer), 562
get_dependencies_folder() (в модулі ахіру), 142
get_destination() (метод ахіру.CsvDataProvider), 213
get_destination() (метод ахіру.DatabaseDataProvider), 230
get_destination() (метод ахіру.DataProvider), 207
get_destination() (метод ахіру.DwgDataProvider), 254
get_destination() (метод ахіру.ExcelDataProvider), 215
get_destination() (метод ахіру.GdalDataProvider), 249
get_destination() (метод ахіру.MifMidDataProvider), 217
get_destination() (метод ахіру.MsSqlDataProvider), 236
get_destination() (метод ахіру.OgrDataProvider), 251
get_destination() (метод ахіру.OracleDataProvider), 233
get_destination() (метод ахіру.PanoramaDataProvider), 257
get_destination() (метод ахіру.PostgreDataProvider), 227
get_destination() (метод ахіру.RestDataProvider), 242
get_destination() (метод ахіру.ShapeDataProvider), 218
get_destination() (метод ахіру.SqliteDataProvider), 221
get_destination() (метод ахіру.SvgDataProvider), 225
get_destination() (метод ахіру.TabDataProvider), 223
get_destination() (метод ахіру.TmsDataProvider), 239
get_destination() (метод ахіру.WmsDataProvider), 245
get_destination() (метод ахіру.WmtsDataProvider), 247
get_distance() (метод ахіру.Arc), 471
get_distance() (метод ахіру.Ellipse), 459
get_distance() (метод ахіру.Geometry), 321
get_distance() (метод ахіру.GeometryCollection), 385
get_distance() (метод ахіру.Line), 344
get_distance() (метод ахіру.LineString), 357
get_distance() (метод ахіру.MultiLineString), 410
get_distance() (метод ахіру.MultiPoint), 398
get_distance() (метод ахіру.MultiPolygon), 423
get_distance() (метод ахіру.Point), 332
get_distance() (метод ахіру.Polygon), 369
get_distance() (метод ахіру.Rectangle), 435
get_distance() (метод ахіру.RoundRectangle), 447
get_distance() (метод ахіру.Text), 485
get_gcps() (метод ахіру.Raster), 533
get_interval_value() (метод ахіру.RangeThematicLayer), 586
get_length() (метод ахіру.Arc), 471
get_length() (метод ахіру.Ellipse), 459
get_length() (метод ахіру.Geometry), 321
get_length() (метод ахіру.GeometryCollection), 385
get_length() (метод ахіру.Line), 345
get_length() (метод ахіру.LineString), 357
get_length() (метод ахіру.MultiLineString), 411
get_length() (метод ахіру.MultiPoint), 398
get_length() (метод ахіру.MultiPolygon), 424
get_length() (метод ахіру.Point), 333
get_length() (метод ахіру.Polygon), 370
get_length() (метод ахіру.Rectangle), 436
get_length() (метод ахіру.RoundRectangle), 447
get_length() (метод ахіру.Text), 485
get_line_direction() (метод ахіру.LineString), 358
get_line_direction() (метод ахіру.ListPoint), 375
get_line_direction() (метод ахіру.Polygon), 370
get_perimeter() (метод ахіру.Arc), 472
get_perimeter() (метод ахіру.Ellipse), 460
get_perimeter() (метод ахіру.Geometry), 322
get_perimeter() (метод ахіру.GeometryCollection), 386
get_perimeter() (метод ахіру.Line), 346
get_perimeter() (метод ахіру.LineString), 358
get_perimeter() (метод ахіру.MultiLineString), 412
get_perimeter() (метод ахіру.MultiPoint), 399
get_perimeter() (метод ахіру.MultiPolygon), 424
get_perimeter() (метод ахіру.Point), 334
get_perimeter() (метод ахіру.Polygon), 370
get_perimeter() (метод ахіру.Rectangle), 436
get_perimeter() (метод ахіру.RoundRectangle), 448
get_perimeter() (метод ахіру.Text), 486
get_plugin_data_dir() (метод ахіру.Plugin), 131
get_printer() (метод ахіру.ReportView), 666
get_select_rect() (метод ахіру.ViewTool), 637
get_source() (метод ахіру.CsvDataProvider), 213
get_source() (метод ахіру.DatabaseDataProvider), 231

get_source() (метод axipy.DataProvider), 207
 get_source() (метод axipy.DwgDataProvider), 254
 get_source() (метод axipy.ExcelDataProvider), 215
 get_source() (метод axipy.GdalDataProvider), 249
 get_source() (метод axipy.MifMidDataProvider), 217
 get_source() (метод axipy.MsSqlDataProvider), 237
 get_source() (метод axipy.OgrDataProvider), 252
 get_source() (метод axipy.OracleDataProvider), 234
 get_source() (метод axipy.PanoramaDataProvider), 258
 get_source() (метод axipy.PostgreDataProvider), 228
 get_source() (метод axipy.RestDataProvider), 242
 get_source() (метод axipy.ShapeDataProvider), 219
 get_source() (метод axipy.SqliteDataProvider), 221
 get_source() (метод axipy.SvgDataProvider), 226
 get_source() (метод axipy.TabDataProvider), 223
 get_source() (метод axipy.TmsDataProvider), 239
 get_source() (метод axipy.WmsDataProvider), 245
 get_source() (метод axipy.WmtsDataProvider), 247
 get_style() (метод axipy.BarThematicLayer), 598
 get_style() (метод axipy.IndividualThematicLayer), 608
 get_style() (метод axipy.PieThematicLayer), 592
 get_style() (метод axipy.RangeThematicLayer), 586
 get_style() (метод axipy.StyledByIndexThematic), 616
 get_value() (метод axipy.IndividualThematicLayer), 608
 global_parent (axipy.ViewManager property), 679
 grayscale (axipy.RasterLayer property), 558
 group() (метод axipy.ListLayers), 551

H

handleEvent() (метод axipy.ViewTool), 638
 has_geometry() (метод axipy.Feature), 304
 has_shadow (axipy.PointFontStyle property), 507
 has_style() (метод axipy.Feature), 304
 HasTables (атрибут axipy.ObserverManager), 537
 height (axipy.Rect property), 189
 height (axipy.Text property), 486
 holes (axipy.Polygon property), 371
 horizontal_pages (axipy.Report property), 617
 horizontalAlign (axipy.Label property), 572
 hotlink (axipy.CosmeticLayer property), 566
 hotlink (axipy.CosmeticTable property), 296
 hotlink (axipy.QueryTable property), 279
 hotlink (axipy.SelectionTable property), 288
 hotlink (axipy.Table property), 271
 hotlink (axipy.VectorLayer property), 562

I

icons (axipy.ActionManager property), 682
 id (axipy.CsvDataProvider property), 213
 id (axipy.DatabaseDataProvider property), 231
 id (axipy.DataProvider property), 207
 id (axipy.DialogTask property), 177
 id (axipy.DwgDataProvider property), 255
 id (axipy.ExcelDataProvider property), 215
 id (axipy.Feature property), 304
 id (axipy.GdalDataProvider property), 250
 id (axipy.MifMidDataProvider property), 217
 id (axipy.MsSqlDataProvider property), 237
 id (axipy.OgrDataProvider property), 252
 id (axipy.OracleDataProvider property), 234
 id (axipy.PanoramaDataProvider property), 258
 id (axipy.PostgreDataProvider property), 228
 id (axipy.RestDataProvider property), 243
 id (axipy.ShapeDataProvider property), 219

id (axipy.SqliteDataProvider property), 221
 id (axipy.SvgDataProvider property), 226
 id (axipy.TabDataProvider property), 223
 id (axipy.Task property), 170
 id (axipy.TmsDataProvider property), 240
 id (axipy.WmsDataProvider property), 245
 id (axipy.WmtsDataProvider property), 248
 ids (axipy.SelectionManager property), 675
 ids() (метод axipy.CustomLabels), 632
 ids() (метод axipy.PluginManager), 133
 IGNORE (атрибут axipy.ViewTool), 637
 index_by_name() (метод axipy.Schema), 307
 IndividualThematicLayer (класс в ахипу), 604
 infinite_progress (axipy.DialogTask property), 177
 info() (метод axipy.PluginManager), 133
 Information (атрибут axipy.Notifications), 692
 init_axioma() (в модуле ахипу), 129
 insert() (метод axipy.CosmeticTable), 297
 insert() (метод axipy.ListLayers), 552
 insert() (метод axipy.ListPoint), 375
 insert() (метод axipy.QueryTable), 280
 insert() (метод axipy.Schema), 307
 insert() (метод axipy.SelectionTable), 289
 insert() (метод axipy.Table), 272
 integer() (статический метод axipy.Attribute), 310
 intersected() (метод axipy.Rect), 189
 intersection() (метод axipy.Arc), 473
 intersection() (метод axipy.Ellipse), 461
 intersection() (метод axipy.Geometry), 322
 intersection() (метод axipy.GeometryCollection), 386
 intersection() (метод axipy.Line), 346
 intersection() (метод axipy.LineString), 358
 intersection() (метод axipy.MultiLineString), 412
 intersection() (метод axipy.MultiPoint), 399
 intersection() (метод axipy.MultiPolygon), 425
 intersection() (метод axipy.Point), 334
 intersection() (метод axipy.Polygon), 371
 intersection() (метод axipy.Rectangle), 437
 intersection() (метод axipy.RoundRectangle), 449
 intersection() (метод axipy.Text), 487
 intersects() (метод axipy.Arc), 473
 intersects() (метод axipy.Ellipse), 461
 intersects() (метод axipy.Geometry), 322
 intersects() (метод axipy.GeometryCollection), 386
 intersects() (метод axipy.GeometryReportItem), 621
 intersects() (метод axipy.LegendReportItem), 628
 intersects() (метод axipy.Line), 346
 intersects() (метод axipy.LineString), 358
 intersects() (метод axipy.MapReportItem), 624
 intersects() (метод axipy.MultiLineString), 412
 intersects() (метод axipy.MultiPoint), 399
 intersects() (метод axipy.MultiPolygon), 425
 intersects() (метод axipy.Point), 334
 intersects() (метод axipy.Polygon), 371
 intersects() (метод axipy.RasterReportItem), 625
 intersects() (метод axipy.Rect), 189
 intersects() (метод axipy.Rectangle), 437
 intersects() (метод axipy.ReportItem), 620
 intersects() (метод axipy.RoundRectangle), 449
 intersects() (метод axipy.ScaleBarReportItem), 630
 intersects() (метод axipy.TableReportItem), 627
 intersects() (метод axipy.Text), 487
 inv_flattening (axipy.CoordSystem property), 157
 is_active (axipy.ViewTool property), 638
 is_canceled (axipy.DialogTask property), 177
 is_canceled (axipy.Task property), 170
 is_editable (axipy.CosmeticTable property), 297
 is_editable (axipy.QueryTable property), 280

is_editable (axipy.SelectionTable property), 289
 is_editable (axipy.Table property), 272
 is_empty (axipy.Rect property), 189
 is_equal() (метод axipy.CoordSystem), 157
 is_loaded() (метод axipy.PluginManager), 133
 is_modified (axipy.CosmeticTable property), 297
 is_modified (axipy.DrawableView property), 653
 is_modified (axipy.MapView property), 660
 is_modified (axipy.QueryTable property), 280
 is_modified (axipy.ReportView property), 666
 is_modified (axipy.SelectionTable property), 289
 is_modified (axipy.Table property), 272
 is_snapped() (метод axipy.ViewTool), 638
 is_spatial (axipy.CosmeticTable property), 297
 is_spatial (axipy.DataObject property), 267
 is_spatial (axipy.QueryTable property), 280
 is_spatial (axipy.Raster property), 533
 is_spatial (axipy.RasteredTable property), 542
 is_spatial (axipy.SelectionTable property), 289
 is_spatial (axipy.Table property), 272
 is_square (axipy.BoundingRectDialog property), 686
 is_supported_coordsystem() (метод axipy.PanoramaDataProvider), 258
 is_temporary (axipy.CosmeticTable property), 297
 is_temporary (axipy.QueryTable property), 280
 is_temporary (axipy.SelectionTable property), 289
 is_temporary (axipy.Table property), 272
 is_toggled (axipy.ToolButton property), 698
 is_valid (axipy.Arc property), 473
 is_valid (axipy.BarThematicLayer property), 599
 is_valid (axipy.CosmeticLayer property), 567
 is_valid (axipy.CosmeticTable property), 297
 is_valid (axipy.DataObject property), 267
 is_valid (axipy.DensityThematicLayer property), 612
 is_valid (axipy.Ellipse property), 461
 is_valid (axipy.Geometry property), 322
 is_valid (axipy.GeometryCollection property), 386
 is_valid (axipy.IndividualThematicLayer property), 608
 is_valid (axipy.Layer property), 555
 is_valid (axipy.Line property), 346
 is_valid (axipy.LineString property), 358
 is_valid (axipy.MainWindow property), 149
 is_valid (axipy.MultiLineString property), 412
 is_valid (axipy.MultiPoint property), 399
 is_valid (axipy.MultiPolygon property), 425
 is_valid (axipy.PieThematicLayer property), 592
 is_valid (axipy.Point property), 334
 is_valid (axipy.Polygon property), 371
 is_valid (axipy.QueryTable property), 280
 is_valid (axipy.RangeThematicLayer property), 586
 is_valid (axipy.Raster property), 533
 is_valid (axipy.RasteredTable property), 543
 is_valid (axipy.RasterLayer property), 559
 is_valid (axipy.Rect property), 189
 is_valid (axipy.Rectangle property), 437
 is_valid (axipy.RoundRectangle property), 449
 is_valid (axipy.SelectionTable property), 289
 is_valid (axipy.SymbolThematicLayer property), 603
 is_valid (axipy.Table property), 272
 is_valid (axipy.Text property), 487
 is_valid (axipy.VectorLayer property), 563
 is_valid_reason (axipy.Arc property), 473
 is_valid_reason (axipy.Ellipse property), 461
 is_valid_reason (axipy.Geometry property), 322
 is_valid_reason (axipy.GeometryCollection property), 386
 is_valid_reason (axipy.Line property), 346
 is_valid_reason (axipy.LineString property), 358

is_valid_reason (axipy.MultiLineString property), 412
 is_valid_reason (axipy.MultiPoint property), 399
 is_valid_reason (axipy.MultiPolygon property), 425
 is_valid_reason (axipy.Point property), 334
 is_valid_reason (axipy.Polygon property), 371
 is_valid_reason (axipy.Rectangle property), 437
 is_valid_reason (axipy.RoundRectangle property), 449
 is_valid_reason (axipy.Text property), 487
 isStacked (axipy.BarThematicLayer property), 599
 italic (axipy.TextStyle property), 526
 items (axipy.Legend property), 578
 items (axipy.Report property), 617
 items() (метод класса axipy.CurrentSettings), 139
 items() (метод axipy.ActionManager), 682
 items() (метод axipy.CosmeticTable), 297
 items() (метод axipy.Feature), 304
 items() (метод axipy.ObserverManager), 538
 items() (метод axipy.QueryTable), 280
 items() (метод axipy.RasteredTable), 543
 items() (метод axipy.SelectionTable), 289
 items() (метод axipy.Table), 272
 items() (метод axipy.TaskManager), 181
 itemsByIds() (метод axipy.CosmeticTable), 297
 itemsByIds() (метод axipy.QueryTable), 280
 itemsByIds() (метод axipy.SelectionTable), 289
 itemsByIds() (метод axipy.Table), 272
 itemsInObject() (метод axipy.CosmeticTable), 298
 itemsInObject() (метод axipy.QueryTable), 281
 itemsInObject() (метод axipy.SelectionTable), 290
 itemsInObject() (метод axipy.Table), 273
 itemsInRect() (метод axipy.CosmeticTable), 298
 itemsInRect() (метод axipy.QueryTable), 281
 itemsInRect() (метод axipy.SelectionTable), 290
 itemsInRect() (метод axipy.Table), 273

K

keyPressEvent() (метод axipy.ViewTool), 638
 keyReleaseEvent() (метод axipy.ViewTool), 638
 keys() (метод класса axipy.CurrentSettings), 139
 keys() (метод axipy.ActionManager), 682
 keys() (метод axipy.Feature), 304
 keys() (метод axipy.ObserverManager), 538
 keys() (метод axipy.TaskManager), 181

L

Label (класс в axipy), 570
 label (axipy.CosmeticLayer property), 567
 label (axipy.VectorLayer property), 563
 LabelAreaInterior (класс в axipy.render), 574
 LabelAreaPosition (класс в axipy.render), 573
 LabelBackgroundType (класс в axipy.render), 573
 LabelHorizontalAlign (класс в axipy.render), 574
 LabelLayout (класс в axipy), 574
 LabelLayoutPosition (класс в axipy), 575
 LabelLinePosition (класс в axipy.render), 573
 LabelOverlap (класс в axipy.render), 573
 Language (атрибут axipy.CurrentSettings), 137
 large() (статический метод axipy.Attribute), 310
 LastNameFilter (атрибут axipy.CurrentSettings), 137
 LastOpenPath (атрибут axipy.CurrentSettings), 137
 LastPathWorkspace (атрибут axipy.CurrentSettings), 137
 LastSavePath (атрибут axipy.CurrentSettings), 137
 lat_lon (axipy.CoordSystem property), 158
 Layer (класс в axipy), 553
 layer_controls (axipy.ViewManager property), 679
 LayerControlWidget (класс в axipy), 688

layers (axipy.Map property), 548
 layers (axipy.RasteredTable property), 543
 LEFT (атрибут axipy.BarThematicLayer), 597
 LEFT (атрибут axipy.OrientationThematic), 615
 LEFT (атрибут axipy.PieThematicLayer), 591
 LEFT_DOWN (атрибут axipy.BarThematicLayer), 597
 LEFT_DOWN (атрибут axipy.OrientationThematic), 615
 LEFT_DOWN (атрибут axipy.PieThematicLayer), 591
 LEFT_UP (атрибут axipy.BarThematicLayer), 597
 LEFT_UP (атрибут axipy.OrientationThematic), 615
 LEFT_UP (атрибут axipy.PieThematicLayer), 591
 Legend (класс в ахипу), 576
 legend (axipy.LegendReportItem property), 629
 LegendItem (класс в ахипу), 579
 LegendReportItem (класс в ахипу), 628
 legends (axipy.LegendView property), 671
 LegendView (класс в ахипу), 670
 legendviews (axipy.ViewManager property), 679
 length (axipy.Attribute property), 310
 Line (класс в ахипу), 337
 line (axipy.CollectionStyle property), 530
 LINEAR (атрибут axipy.AllocationThematic), 613
 LINEAR (атрибут axipy.BarThematicLayer), 597
 LINEAR (атрибут axipy.PieThematicLayer), 591
 LinearUnit (класс в ахипу), 161
 LinearUnits (класс в ахипу), 160
 LineCapStyle (класс в ахипу), 490
 LineDirection (класс в ахипу), 490
 LineJoinStyle (класс в ахипу), 490
 lineKeepDirection (axipy.Label property), 572
 lineLayout (axipy.Label property), 572
 linePosition (axipy.Label property), 572
 linesDirectionVisible (axipy.CosmeticLayer property), 567
 linesDirectionVisible (axipy.VectorLayer property), 563
 LineString (класс в ахипу), 349
 LineStyle (класс в ахипу), 512
 list_widget (axipy.DataManagerWidget property), 690
 ListHoles (класс в ахипу), 375
 ListLayers (класс в ахипу), 550
 ListLegend (класс в ахипу), 673
 ListLegendItems (класс в ахипу), 579
 ListPoint (класс в ахипу), 374
 ListThematic (класс в ахипу), 568
 load() (метод axipy.Plugin), 131
 load() (метод axipy.PluginManager), 133
 load() (метод axipy.ViewTool), 638
 load_file() (метод axipy.Workspace), 683
 load_string() (метод axipy.Workspace), 683
 loaded (axipy.PluginManager property), 133
 loaded_providers() (метод axipy.ProviderManager), 199
 LoadLastWorkspace (атрибут axipy.CurrentSettings), 137
 localized_name (axipy.AreaUnit property), 164
 localized_name (axipy.LinearUnit property), 162
 LOG10 (атрибут axipy.AllocationThematic), 614
 LOG10 (атрибут axipy.BarThematicLayer), 597
 LOG10 (атрибут axipy.PieThematicLayer), 591
 logFile (атрибут axipy.ExportParameters), 212

M

mainwindow (в модуле ахипу), 147
 MainWindow (класс в ахипу), 147
 mainwindow_activated (axipy.ViewManager property), 679

majorSemiAxis (axipy.Ellipse property), 461
 make_acceptable() (метод axipy.ActiveToolPanel), 642
 make_custom() (метод axipy.ActiveToolPanel), 643
 MANUAL (атрибут axipy.RangeThematicLayer), 585
 Map (класс в ахипу), 545
 map (axipy.MapView property), 660
 map() (метод axipy.MapReportItem), 624
 map_rect (axipy.MapReportItem property), 624
 mapCatalog (атрибут axipy.ExportParameters), 212
 MapReportItem (класс в ахипу), 622
 MapView (класс в ахипу), 655
 mapview_activated (axipy.LayerControlWidget property), 689
 mapviews (axipy.ViewManager property), 679
 max (axipy.DialogTask property), 177
 max (axipy.Task property), 171
 max_zoom (axipy.BarThematicLayer property), 599
 max_zoom (axipy.CosmeticLayer property), 567
 max_zoom (axipy.DensityThematicLayer property), 612
 max_zoom (axipy.IndividualThematicLayer property), 608
 max_zoom (axipy.Layer property), 555
 max_zoom (axipy.PieThematicLayer property), 593
 max_zoom (axipy.RangeThematicLayer property), 587
 max_zoom (axipy.RasterLayer property), 559
 max_zoom (axipy.SymbolThematicLayer property), 603
 max_zoom (axipy.VectorLayer property), 563
 maxHeight (axipy.SymbolThematicLayer property), 603
 merge() (метод axipy.Rect), 189
 mesh_size (axipy.ReportView property), 666
 MeshSizeLayout (атрибут axipy.CurrentSettings), 137
 MeshSizeLegend (атрибут axipy.CurrentSettings), 137
 message (axipy.DialogTask property), 177
 message (axipy.Task property), 171
 message_changed (axipy.DialogTask property), 177
 message_changed (axipy.Task property), 171
 mif (axipy.ProviderManager property), 200
 MifMidDataProvider (класс в ахипу), 215
 min (axipy.DialogTask property), 177
 min (axipy.Task property), 171
 min_zoom (axipy.BarThematicLayer property), 599
 min_zoom (axipy.CosmeticLayer property), 567
 min_zoom (axipy.DensityThematicLayer property), 612
 min_zoom (axipy.IndividualThematicLayer property), 608
 min_zoom (axipy.Layer property), 555
 min_zoom (axipy.PieThematicLayer property), 593
 min_zoom (axipy.RangeThematicLayer property), 587
 min_zoom (axipy.RasterLayer property), 559
 min_zoom (axipy.SymbolThematicLayer property), 603
 min_zoom (axipy.VectorLayer property), 563
 minHeight (axipy.SymbolThematicLayer property), 603
 minorSemiAxis (axipy.Ellipse property), 461
 minutes (axipy.AngleCoord property), 195
 mouse_moved (axipy.MapView property), 660
 mouse_moved (axipy.ReportView property), 666
 mouseDoubleClickEvent() (метод axipy.ViewTool), 639
 mouseMoveEvent() (метод axipy.ViewTool), 639
 mousePressEvent() (метод axipy.ViewTool), 639
 mouseReleaseEvent() (метод axipy.ViewTool), 639
 move() (метод axipy.ListLayers), 552
 move() (метод axipy.ListThematic), 569
 mssql (axipy.ProviderManager property), 200
 MsSqlDataProvider (класс в ахипу), 235
 MultiLineString (класс в ахипу), 402
 MultiPoint (класс в ахипу), 389
 MultiPolygon (класс в ахипу), 415

N

name (axipy.Arc property), 473
 name (axipy.AreaUnit property), 164
 name (axipy.Attribute property), 311
 name (axipy.CoordSystem property), 158
 name (axipy.CosmeticTable property), 298
 name (axipy.DataObject property), 267
 name (axipy.DialogTask property), 177
 name (axipy.Ellipse property), 461
 name (axipy.Geometry property), 322
 name (axipy.GeometryCollection property), 386
 name (axipy.Line property), 346
 name (axipy.LinearUnit property), 162
 name (axipy.LineString property), 358
 name (axipy.MultiLineString property), 412
 name (axipy.MultiPoint property), 399
 name (axipy.MultiPolygon property), 425
 name (axipy.Observer property), 539
 name (axipy.Point property), 334
 name (axipy.Polygon property), 371
 name (axipy.QueryTable property), 281
 name (axipy.Raster property), 533
 name (axipy.RasteredTable property), 543
 name (axipy.Rectangle property), 437
 name (axipy.Report property), 617
 name (axipy.RoundRectangle property), 449
 name (axipy.SelectionTable property), 290
 name (axipy.Table property), 273
 name (axipy.Task property), 171
 name (axipy.Text property), 487
 NearlyGeometriesTopology (атрибут axipy.CurrentSettings), 137
 need_redraw (axipy.BarThematicLayer property), 599
 need_redraw (axipy.CosmeticLayer property), 567
 need_redraw (axipy.DensityThematicLayer property), 612
 need_redraw (axipy.IndividualThematicLayer property), 608
 need_redraw (axipy.Layer property), 555
 need_redraw (axipy.Map property), 549
 need_redraw (axipy.PieThematicLayer property), 593
 need_redraw (axipy.RangeThematicLayer property), 587
 need_redraw (axipy.RasterLayer property), 559
 need_redraw (axipy.Report property), 617
 need_redraw (axipy.SymbolThematicLayer property), 603
 need_redraw (axipy.VectorLayer property), 563
 NodesUpdateMode (атрибут axipy.CurrentSettings), 137
 nodesVisible (axipy.CosmeticLayer property), 567
 nodesVisible (axipy.VectorLayer property), 563
 non_earth (axipy.CoordSystem property), 158
 normalize() (метод axipy.Rect), 189
 Notifications (класс в axipy), 692
 NotificationWidget (класс в axipy), 691
 number() (статический метод axipy.Version), 151

O

objects (axipy.DataManager property), 262
 objects (axipy.DataManagerWidget property), 690
 Observer (класс в axipy), 539
 observer (axipy.ActionButton property), 696
 observer (axipy.Button property), 694
 observer (axipy.Separator property), 699
 observer (axipy.ToolButton property), 698
 observer_id (axipy.ActionButton property), 696
 observer_id (axipy.Button property), 694
 observer_id (axipy.Separator property), 699

observer_id (axipy.ToolButton property), 699
 observer_manager (в модуле axipy), 142
 ObserverManager (класс в axipy), 536
 offset (axipy.LabelLayout property), 574
 offset() (метод axipy.DrawableView), 653
 offset() (метод axipy.MapView), 660
 offset() (метод axipy.ReportView), 667
 ogr (axipy.ProviderManager property), 200
 OgrDataProvider (класс в axipy), 250
 on_active_changed() (метод axipy.ViewTool), 639
 opacity (axipy.BarThematicLayer property), 599
 opacity (axipy.CosmeticLayer property), 567
 opacity (axipy.DensityThematicLayer property), 613
 opacity (axipy.IndividualThematicLayer property), 609
 opacity (axipy.Label property), 572
 opacity (axipy.Layer property), 556
 opacity (axipy.PieThematicLayer property), 593
 opacity (axipy.RangeThematicLayer property), 587
 opacity (axipy.RasterLayer property), 559
 opacity (axipy.SymbolThematicLayer property), 603
 opacity (axipy.VectorLayer property), 563
 open() (метод axipy.CsvDataProvider), 213
 open() (метод axipy.DatabaseDataProvider), 231
 open() (метод axipy.DataProvider), 207
 open() (метод axipy.DwgDataProvider), 255
 open() (метод axipy.ExcelDataProvider), 215
 open() (метод axipy.GdalDataProvider), 250
 open() (метод axipy.MifMidDataProvider), 217
 open() (метод axipy.MsSqlDataProvider), 237
 open() (метод axipy.OgrDataProvider), 252
 open() (метод axipy.OracleDataProvider), 234
 open() (метод axipy.PanoramaDataProvider), 258
 open() (метод axipy.PostgreDataProvider), 228
 open() (метод axipy.ProviderManager), 200
 open() (метод axipy.RestDataProvider), 243
 open() (метод axipy.ShapeDataProvider), 219
 open() (метод axipy.Source), 208
 open() (метод axipy.SQLiteDataProvider), 221
 open() (метод axipy.SvgDataProvider), 226
 open() (метод axipy.TabDataProvider), 224
 open() (метод axipy.TmsDataProvider), 240
 open() (метод axipy.WmsDataProvider), 245
 open() (метод axipy.WmtsDataProvider), 248
 open_file_dialog() (в модуле axipy), 140
 open_files_dialog() (в модуле axipy), 140
 open_hidden() (метод axipy.ProviderManager), 203
 open_temporary() (метод axipy.ShapeDataProvider), 219
 openfile() (метод axipy.ProviderManager), 203
 OpenMode (класс в axipy), 259
 oracle (axipy.ProviderManager property), 203
 OracleDataProvider (класс в axipy), 232
 OrientationThematic (класс в axipy), 614
 orientationType (axipy.BarThematicLayer property), 599
 orientationType (axipy.OrientationThematic property), 615
 orientationType (axipy.PieThematicLayer property), 593
 overhang (axipy.Label property), 572
 overlaps() (метод axipy.Arc), 473
 overlaps() (метод axipy.Ellipse), 461
 overlaps() (метод axipy.Geometry), 322
 overlaps() (метод axipy.GeometryCollection), 386
 overlaps() (метод axipy.Line), 346
 overlaps() (метод axipy.LineString), 358
 overlaps() (метод axipy.MultiLineString), 412
 overlaps() (метод axipy.MultiPoint), 399

overlaps() (метод axipy.MultiPolygon), 425
 overlaps() (метод axipy.Point), 334
 overlaps() (метод axipy.Polygon), 371
 overlaps() (метод axipy.Rectangle), 437
 overlaps() (метод axipy.RoundRectangle), 449
 overlaps() (метод axipy.Text), 487
 overrideStyle (axipy.CosmeticLayer property), 567
 overrideStyle (axipy.VectorLayer property), 563

P

page_size (axipy.Report property), 618
 paintEvent() (метод axipy.ViewTool), 639
 panel_was_closed
 (axipy.AxipyActiveToolPanelHandlerBase property), 644
 panorama (axipy.ProviderManager property), 203
 PanoramaDataProvider (класс в axipy), 256
 password (axipy.PasswordDialog property), 685
 PasswordDialog (класс в axipy), 685
 pattern (axipy.FillStyle property), 518
 pattern (axipy.LineStyle property), 515
 PenCatalog (атрибут axipy.CurrentSettings), 137
 PhysicalScreenSize (атрибут axipy.CurrentSettings), 137
 PieThematicLayer (класс в axipy), 589
 placementPolicy (axipy.Label property), 572
 Plugin (класс в axipy), 130
 plugin_dir (axipy.Plugin property), 131
 plugin_manager (в модуле axipy), 142
 PluginInfo (класс в axipy), 134
 PluginManager (класс в axipy), 132
 Pnt (класс в axipy), 183
 Point (класс в axipy), 325
 point (axipy.CollectionStyle property), 530
 point_by_azimuth() (статический метод axipy.Arc), 473
 point_by_azimuth() (статический метод axipy.Ellipse), 461
 point_by_azimuth() (статический метод axipy.Geometry), 322
 point_by_azimuth() (статический метод axipy.GeometryCollection), 386
 point_by_azimuth() (статический метод axipy.Line), 346
 point_by_azimuth() (статический метод axipy.LineString), 359
 point_by_azimuth() (статический метод axipy.MultiLineString), 412
 point_by_azimuth() (статический метод axipy.MultiPoint), 399
 point_by_azimuth() (статический метод axipy.MultiPolygon), 425
 point_by_azimuth() (статический метод axipy.Point), 334
 point_by_azimuth() (статический метод axipy.Polygon), 371
 point_by_azimuth() (статический метод axipy.Rectangle), 437
 point_by_azimuth() (статический метод axipy.RoundRectangle), 449
 point_by_azimuth() (статический метод axipy.Text), 487
 PointCompatStyle (класс в axipy), 499
 PointFontStyle (класс в axipy), 503
 PointFontStyle.FontStyleTypes (класс в axipy), 505
 pointForMaximum (axipy.DensityThematicLayer property), 613
 pointLayout (axipy.Label property), 572
 PointPictureStyle (класс в axipy), 508
 points (axipy.LineString property), 359
 points (axipy.Polygon property), 372
 PointStyle (класс в axipy), 495
 Polygon (класс в axipy), 361
 polygon (axipy.CollectionStyle property), 531
 PolygonStyle (класс в axipy), 519
 pop() (метод axipy.ListPoint), 375
 Position (класс в axipy), 700
 position (axipy.CustomLabelProperties property), 576
 position (axipy.DrawableView property), 653
 position (axipy.LabelLayout property), 574
 position (axipy.Legend property), 578
 position (axipy.LegendView property), 671
 position (axipy.MapView property), 660
 position (axipy.ReportView property), 667
 position (axipy.TableView property), 650
 position (axipy.View property), 648
 position_align (axipy.CustomLabelProperties property), 576
 postgre (axipy.ProviderManager property), 203
 PostgreDataProvider (класс в axipy), 226
 precision (axipy.Attribute property), 311
 preserve_aspect_ratio (axipy.RasterReportItem property), 625
 PreserveScaleMap (атрибут axipy.CurrentSettings), 137
 prj (axipy.CoordSystem property), 158
 proj (axipy.CoordSystem property), 158
 proj_transform_definition() (метод класса axipy.CoordTransformer), 159
 prompt_coordsystem() (в модуле axipy), 146
 prompt_float() (в модуле axipy), 144
 prompt_int() (в модуле axipy), 144
 prompt_item() (в модуле axipy), 145
 prompt_rect() (в модуле axipy), 146
 prompt_string() (в модуле axipy), 144
 properties (axipy.CosmeticTable property), 298
 properties (axipy.DataObject property), 267
 properties (axipy.QueryTable property), 281
 properties (axipy.Raster property), 533
 properties (axipy.RasteredTable property), 543
 properties (axipy.SelectionTable property), 291
 properties (axipy.Table property), 273
 provider (axipy.CosmeticTable property), 299
 provider (axipy.DataObject property), 267
 provider (axipy.QueryTable property), 282
 provider (axipy.Raster property), 533
 provider (axipy.RasteredTable property), 543
 provider (axipy.SelectionTable property), 291
 provider (axipy.Table property), 274
 provider_manager (в модуле axipy), 142
 ProviderManager (класс в axipy), 197
 providers() (метод axipy.ProviderManager), 203
 push() (статический метод axipy.Notifications), 692
 PythonConsoleWidget (класс в axipy), 691

Q

qt_format() (статический метод axipy.Version), 151
 qtFormat() (статический метод axipy.Version), 151
 query() (метод axipy.DataManager), 262
 query_hidden() (метод axipy.DataManager), 263
 QueryTable (класс в axipy), 275
 quick_toolbar (axipy.MainWindow property), 149

R

- ul style="list-style-type: none; padding-left: 0;">
- raise_if_canceled() (метод ахипу.DialogTask), 177
- raise_if_canceled() (метод ахипу.Task), 171
- range (ахипу.DialogTask property), 177
- range (ахипу.Task property), 171
- range_changed (ахипу.DialogTask property), 178
- range_changed (ахипу.Task property), 171
- rangeEnabled (ахипу.Label property), 572
- rangeMax (ахипу.Label property), 572
- rangeMin (ахипу.Label property), 572
- ranges (ахипу.RangeThematicLayer property), 587
- RangeThematicLayer (класс в ахипу), 582
- Raster (класс в ахипу), 531
- RasteredTable (класс в ахипу), 541
- RasterLayer (класс в ахипу), 556
- RasterReportItem (класс в ахипу), 624
- read_contents() (метод ахипу.ProviderManager), 203
- readOnly (ахипу.Attribute property), 311
- ReallocateThematicColor (класс в ахипу), 581
- Rect (класс в ахипу), 186
- rect (ахипу.BoundingRectDialog property), 686
- rect (ахипу.Context property), 631
- rect (ахипу.CoordSystem property), 158
- rect (ахипу.GeometryReportItem property), 621
- rect (ахипу.LegendReportItem property), 629
- rect (ахипу.MapReportItem property), 624
- rect (ахипу.RasterReportItem property), 625
- rect (ахипу.ReportItem property), 620
- rect (ахипу.ScaleBarReportItem property), 630
- rect (ахипу.TableReportItem property), 627
- rect_as_polygon (ахипу.Text property), 487
- Rectangle (класс в ахипу), 428
- redo() (метод ахипу.CosmeticTable), 299
- redo() (метод ахипу.DrawableView), 653
- redo() (метод ахипу.MapView), 661
- redo() (метод ахипу.QueryTable), 282
- redo() (метод ахипу.ReportView), 667
- redo() (метод ахипу.SelectionTable), 291
- redo() (метод ахипу.Table), 274
- redraw() (метод ахипу.ViewTool), 640
- refresh() (метод ахипу.Legend), 578
- refreshValues() (метод ахипу.TableReportItem), 627
- register() (в модуле ахипу), 535
- relate() (метод ахипу.Arc), 473
- relate() (метод ахипу.Ellipse), 461
- relate() (метод ахипу.Geometry), 323
- relate() (метод ахипу.GeometryCollection), 387
- relate() (метод ахипу.Line), 347
- relate() (метод ахипу.LineString), 359
- relate() (метод ахипу.MultiLineString), 413
- relate() (метод ахипу.MultiPoint), 400
- relate() (метод ахипу.MultiPolygon), 425
- relate() (метод ахипу.Point), 335
- relate() (метод ахипу.Polygon), 372
- relate() (метод ахипу.Rectangle), 437
- relate() (метод ахипу.RoundRectangle), 449
- relate() (метод ахипу.Text), 487
- remove() (метод ахипу.ActionButton), 696
- remove() (метод ахипу.Button), 694
- remove() (метод ахипу.CosmeticTable), 299
- remove() (метод ахипу.DataManager), 263
- remove() (метод ахипу.GeometryCollection), 387
- remove() (метод ахипу.ListHoles), 376
- remove() (метод ахипу.ListLayers), 552
- remove() (метод ахипу.ListPoint), 375
- remove() (метод ахипу.ListThematic), 569
- remove() (метод ахипу.MultiLineString), 413
- remove() (метод ахипу.MultiPoint), 400
- remove() (метод ахипу.MultiPolygon), 426
- remove() (метод ахипу.ObserverManager), 538
- remove() (метод ахипу.QueryTable), 282
- remove() (метод ахипу.ReportItems), 619
- remove() (метод ахипу.SelectionManager), 675
- remove() (метод ахипу.SelectionTable), 291
- remove() (метод ахипу.Separator), 700
- remove() (метод ахипу.SystemActionButton), 697
- remove() (метод ахипу.Table), 274
- remove() (метод ахипу.ToolButton), 699
- remove_all() (метод ахипу.DataManager), 263
- remove_dock_widget() (метод ахипу.MainWindow), 149
- remove_table_files() (метод ахипу.TabDataProvider), 224
- removed (ахипу.DataManager property), 263
- removed (ахипу.TaskManager property), 181
- rename_table_files() (метод ахипу.TabDataProvider), 224
- RenameDataObjectFromTab (атрибут ахипу.CurrentSettings), 137
- renditonColumnName (атрибут ахипу.ExportParameters), 212
- Report (класс в ахипу), 616
- report (ахипу.ReportView property), 667
- ReportItem (класс в ахипу), 619
- ReportItems (класс в ахипу), 618
- ReportView (класс в ахипу), 663
- reportviews (ахипу.ViewManager property), 680
- reproject() (метод ахипу.Arc), 473
- reproject() (метод ахипу.Ellipse), 461
- reproject() (метод ахипу.Geometry), 323
- reproject() (метод ахипу.GeometryCollection), 387
- reproject() (метод ахипу.Line), 347
- reproject() (метод ахипу.LineString), 359
- reproject() (метод ахипу.MultiLineString), 413
- reproject() (метод ахипу.MultiPoint), 400
- reproject() (метод ахипу.MultiPolygon), 426
- reproject() (метод ахипу.Point), 335
- reproject() (метод ахипу.Polygon), 372
- reproject() (метод ахипу.Rectangle), 438
- reproject() (метод ахипу.RoundRectangle), 449
- reproject() (метод ахипу.Text), 487
- Resample (класс в ахипу), 534
- reset() (статический метод ахипу.CurrentSettings), 139
- reset() (статический метод ахипу.ViewTool), 640
- reset_parent() (метод ахипу.DrawableView), 653
- reset_parent() (метод ахипу.LegendView), 671
- reset_parent() (метод ахипу.MapView), 661
- reset_parent() (метод ахипу.ReportView), 667
- reset_parent() (метод ахипу.TableView), 650
- reset_parent() (метод ахипу.View), 648
- rest (ахипу.ProviderManager property), 204
- RestDataProvider (класс в ахипу), 241
- result (ахипу.DialogTask property), 178
- result (ахипу.Task property), 171
- RIGHT (атрибут ахипу.BarThematicLayer), 597
- RIGHT (атрибут ахипу.OrientationThematic), 615
- RIGHT (атрибут ахипу.PieThematicLayer), 591
- RIGHT_DOWN (атрибут ахипу.BarThematicLayer), 597
- RIGHT_DOWN (атрибут ахипу.OrientationThematic), 615
- RIGHT_DOWN (атрибут ахипу.PieThematicLayer), 591
- RIGHT_UP (атрибут ахипу.BarThematicLayer), 597
- RIGHT_UP (атрибут ахипу.OrientationThematic), 615
- RIGHT_UP (атрибут ахипу.PieThematicLayer), 591
- rollback() (метод ахипу.CosmeticTable), 299
- rollback() (метод ахипу.QueryTable), 282
- rollback() (метод ахипу.SelectionTable), 291

rollback() (метод axipy.Table), 274
 rotate() (метод axipy.Arc), 473
 rotate() (метод axipy.Ellipse), 461
 rotate() (метод axipy.Geometry), 323
 rotate() (метод axipy.GeometryCollection), 387
 rotate() (метод axipy.Line), 347
 rotate() (метод axipy.LineString), 359
 rotate() (метод axipy.MultiLineString), 413
 rotate() (метод axipy.MultiPoint), 400
 rotate() (метод axipy.MultiPolygon), 426
 rotate() (метод axipy.Point), 335
 rotate() (метод axipy.Polygon), 372
 rotate() (метод axipy.Rectangle), 438
 rotate() (метод axipy.RoundRectangle), 449
 rotate() (метод axipy.Text), 487
 rotation (axipy.PointFontStyle property), 507
 RoundRectangle (класс в axipy), 440
 row_count (axipy.TableReportItem property), 627
 row_from (axipy.TableReportItem property), 627
 RuLerColorLine (атрибут axipy.CurrentSettings), 137
 run_and_get() (метод axipy.DialogTask), 178
 run_and_get() (метод axipy.Task), 171
 run_and_get() (метод axipy.TaskManager), 181
 run_in_gui() (в модуле axipy), 142
 run_in_gui() (метод axipy.TaskManager), 182

S

save_file() (метод axipy.Workspace), 684
 save_file_dialog() (в модуле axipy), 141
 save_string() (метод axipy.Workspace), 684
 save_template() (метод axipy.ReportView), 668
 SaveAsToOriginalFileFolder (атрибут axipy.CurrentSettings), 138
 scale (axipy.MapReportItem property), 624
 scale (axipy.MapView property), 661
 scale() (метод axipy.Arc), 474
 scale() (метод axipy.Ellipse), 462
 scale() (метод axipy.Geometry), 323
 scale() (метод axipy.GeometryCollection), 387
 scale() (метод axipy.Line), 347
 scale() (метод axipy.LineString), 359
 scale() (метод axipy.MultiLineString), 413
 scale() (метод axipy.MultiPoint), 400
 scale() (метод axipy.MultiPolygon), 426
 scale() (метод axipy.Point), 335
 scale() (метод axipy.Polygon), 372
 scale() (метод axipy.Rectangle), 438
 scale() (метод axipy.RoundRectangle), 450
 scale() (метод axipy.Text), 488
 scale_with_center() (метод axipy.DrawableView), 654
 scale_with_center() (метод axipy.MapView), 661
 scale_with_center() (метод axipy.ReportView), 668
 ScaleBarReportItem (класс в axipy), 629
 scene_changed (axipy.DrawableView property), 654
 scene_changed (axipy.MapView property), 661
 scene_changed (axipy.ReportView property), 668
 scene_rect (axipy.MapView property), 661
 scene_to_device_transform (axipy.DrawableView property), 654
 scene_to_device_transform (axipy.MapView property), 662
 scene_to_device_transform (axipy.Raster property), 533
 scene_to_device_transform (axipy.ReportView property), 668
 Schema (класс в axipy), 305
 schema (axipy.CosmeticTable property), 299

schema (axipy.QueryTable property), 282
 schema (axipy.RasteredTable property), 543
 schema (axipy.SelectionTable property), 291
 schema (axipy.Table property), 274
 schema_changed (axipy.CosmeticTable property), 299
 schema_changed (axipy.QueryTable property), 282
 schema_changed (axipy.SelectionTable property), 291
 schema_changed (axipy.Table property), 274
 seconds (axipy.AngleCoord property), 195
 segments() (статический метод axipy.Version), 151
 select_folder_dialog() (в модуле axipy), 141
 select_style_dialog() (в модуле axipy), 145
 selectable (axipy.BarThematicLayer property), 599
 selectable (axipy.CosmeticLayer property), 567
 selectable (axipy.DensityThematicLayer property), 613
 selectable (axipy.IndividualThematicLayer property), 609
 selectable (axipy.Layer property), 556
 selectable (axipy.PieThematicLayer property), 593
 selectable (axipy.RangeThematicLayer property), 587
 selectable (axipy.RasterLayer property), 559
 selectable (axipy.SymbolThematicLayer property), 603
 selectable (axipy.VectorLayer property), 563
 SelectByInformationTool (атрибут axipy.CurrentSettings), 138
 selected_layer (axipy.MapView property), 662
 selected_layers (axipy.LayerControlWidget property), 689
 SelectedFilter (класс в axipy), 140
 Selection (атрибут axipy.ObserverManager), 538
 selection (axipy.DataManager property), 263
 selection_changed (axipy.DataManagerWidget property), 690
 selection_changed (axipy.LayerControlWidget property), 689
 selection_manager (в модуле axipy), 142
 SelectionEditable (атрибут axipy.ObserverManager), 538
 SelectionEditableIsSame (атрибут axipy.ObserverManager), 538
 SelectionManager (класс в axipy), 673
 SelectionTable (класс в axipy), 284
 semi_major (axipy.CoordSystem property), 158
 semi_minor (axipy.CoordSystem property), 158
 SensitiveMouse (атрибут axipy.CurrentSettings), 138
 Separator (класс в axipy), 699
 set() (метод axipy.CustomLabels), 632
 set() (метод axipy.SelectionManager), 675
 set_all() (метод axipy.SelectionManager), 676
 set_brush() (метод axipy.PolygonStyle), 521
 set_current() (метод класса axipy.CoordSystem), 158
 set_default_arrow_type() (метод axipy.CustomLabels), 632
 set_default_end_type() (метод axipy.CustomLabels), 632
 set_interval_value() (метод axipy.RangeThematicLayer), 587
 set_line_direction() (метод axipy.LineString), 360
 set_line_direction() (метод axipy.ListPoint), 375
 set_line_direction() (метод axipy.Polygon), 372
 set_observer() (метод axipy.AxipyActiveToolPanelHandlerBase), 644
 set_palette() (метод axipy.DwgDataProvider), 255
 set_panel_title() (метод axipy.AxipyActiveToolPanelHandlerBase), 645
 set_pen() (метод axipy.PolygonStyle), 522
 set_printer() (метод axipy.ReportView), 668
 set_style() (метод axipy.BarThematicLayer), 600

- set_style() (метод axipy.IndividualThematicLayer), 609
- set_style() (метод axipy.PieThematicLayer), 593
- set_style() (метод axipy.RangeThematicLayer), 587
- set_style() (метод axipy.StyledByIndexThematic), 616
- set_widget() (метод axipy.AxipyActiveToolPanelHandlerBase), 645
- set_zoom() (метод axipy.MapView), 662
- set_zoom_and_center() (метод axipy.MapView), 662
- settings (axipy.Plugin property), 131
- shadow (axipy.Label property), 572
- shadow (axipy.TextStyle property), 526
- ShapeDataProvider (класс в axipy), 218
- shift() (метод axipy.Arc), 474
- shift() (метод axipy.Ellipse), 462
- shift() (метод axipy.Geometry), 323
- shift() (метод axipy.GeometryCollection), 387
- shift() (метод axipy.Line), 347
- shift() (метод axipy.LineString), 360
- shift() (метод axipy.MultiLineString), 413
- shift() (метод axipy.MultiPoint), 400
- shift() (метод axipy.MultiPolygon), 426
- shift() (метод axipy.Point), 335
- shift() (метод axipy.Polygon), 372
- shift() (метод axipy.Rectangle), 438
- shift() (метод axipy.RoundRectangle), 450
- shift() (метод axipy.Text), 488
- short() (статический метод axipy.Attribute), 311
- show() (метод axipy.DrawableView), 654
- show() (метод axipy.LegendView), 672
- show() (метод axipy.MapView), 662
- show() (метод axipy.ReportView), 668
- show() (метод axipy.TableView), 651
- show() (метод axipy.View), 649
- show() (статический метод axipy.MainWindow), 149
- show_all() (метод axipy.MapReportItem), 624
- show_all() (метод axipy.MapView), 662
- show_background (axipy.PointPictureStyle property), 512
- show_borders (axipy.ReportView property), 668
- show_dialog() (в модуле axipy), 143
- show_elements_size (axipy.ReportView property), 668
- show_html_url() (метод axipy.MainWindow), 149
- show_mesh (axipy.ReportView property), 668
- show_message() (в модуле axipy), 143
- show_row_number (axipy.TableReportItem property), 627
- show_ruler (axipy.ReportView property), 669
- show_selection() (метод axipy.MapView), 662
- show_type (axipy.DrawableView property), 655
- show_type (axipy.LegendView property), 672
- show_type (axipy.MapView property), 662
- show_type (axipy.ReportView property), 669
- show_type (axipy.TableView property), 651
- show_type (axipy.View property), 649
- showCentroid (axipy.CosmeticLayer property), 567
- showCentroid (axipy.VectorLayer property), 563
- ShowDegreeTypeNumeric (атрибут axipy.CurrentSettings), 138
- ShowDrawingToolTip (атрибут axipy.CurrentSettings), 138
- ShowMapScaleBar (атрибут axipy.CurrentSettings), 138
- ShowMeshLayout (атрибут axipy.CurrentSettings), 138
- ShowMeshLegend (атрибут axipy.CurrentSettings), 138
- ShowScrollOnMapView (атрибут axipy.CurrentSettings), 138
- ShowSplashScreen (атрибут axipy.CurrentSettings), 138
- shp (axipy.ProviderManager property), 204
- SilentCloseWidget (атрибут axipy.CurrentSettings), 138
- size (axipy.DensityThematicLayer property), 613
- size (axipy.PointCompatStyle property), 502
- size (axipy.PointFontStyle property), 508
- size (axipy.PointPictureStyle property), 512
- size (axipy.Raster property), 533
- size (axipy.TextStyle property), 526
- size_for_view() (метод axipy.Text), 488
- snap() (метод axipy.ViewTool), 640
- snap_device() (метод axipy.ViewTool), 640
- snap_mode (axipy.DrawableView property), 655
- snap_mode (axipy.MapView property), 662
- snap_mode (axipy.ReportView property), 669
- snap_to_guidelines (axipy.ReportView property), 669
- snap_to_mesh (axipy.ReportView property), 669
- SnapSensitiveRadius (атрибут axipy.CurrentSettings), 138
- SnapToMeshLayout (атрибут axipy.CurrentSettings), 138
- SnapToMeshLegend (атрибут axipy.CurrentSettings), 138
- Source (класс в axipy), 208
- spacing (axipy.Label property), 572
- spacing (axipy.TextStyle property), 526
- split_by_polyline() (метод axipy.Arc), 474
- split_by_polyline() (метод axipy.Ellipse), 462
- split_by_polyline() (метод axipy.Geometry), 324
- split_by_polyline() (метод axipy.GeometryCollection), 388
- split_by_polyline() (метод axipy.Line), 347
- split_by_polyline() (метод axipy.LineString), 360
- split_by_polyline() (метод axipy.MultiLineString), 413
- split_by_polyline() (метод axipy.MultiPoint), 401
- split_by_polyline() (метод axipy.MultiPolygon), 426
- split_by_polyline() (метод axipy.Point), 335
- split_by_polyline() (метод axipy.Polygon), 373
- split_by_polyline() (метод axipy.Rectangle), 438
- split_by_polyline() (метод axipy.RoundRectangle), 450
- split_by_polyline() (метод axipy.Text), 488
- splitType (axipy.RangeThematicLayer property), 587
- sql_dialect (axipy.DataManager property), 263
- sql_text (axipy.QueryTable property), 283
- sqlite (axipy.ProviderManager property), 204
- SqliteDataProvider (класс в axipy), 220
- SqliteGeometryFormat (класс в axipy), 259
- SQRT (атрибут axipy.AllocationThematic), 614
- SQRT (атрибут axipy.BarThematicLayer), 597
- SQRT (атрибут axipy.PieThematicLayer), 591
- srid (атрибут axipy.ExportParameters), 212
- start() (метод axipy.DialogTask), 178
- start() (метод axipy.Task), 172
- start_number (axipy.TableReportItem property), 627
- start_task() (метод axipy.TaskManager), 182
- startAngle (axipy.Arc property), 474
- startAngle (axipy.PieThematicLayer property), 594
- started (axipy.DialogTask property), 178
- started (axipy.Task property), 172
- startPoint (axipy.Text property), 488
- status (axipy.ClipGeometry property), 633
- status (axipy.DialogTask property), 178
- status (axipy.Task property), 172
- string() (статический метод axipy.Attribute), 311
- string() (статический метод axipy.Version), 151
- Style (класс в axipy), 492
- style (axipy.Feature property), 304

style (axipy.GeometryReportItem property), 622
 style (axipy.LegendItem property), 579
 style (axipy.StyleButton property), 687
 style_caption (axipy.Legend property), 578
 style_changed (axipy.StyleButton property), 687
 style_geometry_type (axipy.IndividualThematicLayer property), 609
 style_geometry_type (axipy.RangeThematicLayer property), 588
 style_subcaption (axipy.Legend property), 578
 style_text (axipy.Legend property), 578
 StyleButton (класс в ахипу), 686
 StyledByIndexThematic (класс в ахипу), 615
 StyleGeometryType (класс в ахипу), 495
 subcaption (axipy.Legend property), 578
 Success (атрибут axipy.Notifications), 692
 suggest_tab_name() (метод axipy.TabFile), 541
 supported_calc_modes (axipy.CoordSystem property), 158
 supported_operations (axipy.CosmeticTable property), 300
 supported_operations (axipy.QueryTable property), 283
 supported_operations (axipy.SelectionTable property), 292
 supported_operations (axipy.Table property), 275
 SupportedOperations (класс в ахипу), 300
 supressDuplicates (axipy.Label property), 573
 svg (axipy.ProviderManager property), 204
 SvgDataProvider (класс в ахипу), 224
 symbol (axipy.PointCompatStyle property), 502
 symbol (axipy.PointFontStyle property), 508
 SymbolCatalog (атрибут axipy.CurrentSettings), 138
 SymbolThematicLayer (класс в ахипу), 600
 symmetric_difference() (метод axipy.Arc), 474
 symmetric_difference() (метод axipy.Ellipse), 462
 symmetric_difference() (метод axipy.Geometry), 324
 symmetric_difference() (метод axipy.GeometryCollection), 388
 symmetric_difference() (метод axipy.Line), 348
 symmetric_difference() (метод axipy.LineString), 360
 symmetric_difference() (метод axipy.MultiLineString), 414
 symmetric_difference() (метод axipy.MultiPoint), 401
 symmetric_difference() (метод axipy.MultiPolygon), 427
 symmetric_difference() (метод axipy.Point), 336
 symmetric_difference() (метод axipy.Polygon), 373
 symmetric_difference() (метод axipy.Rectangle), 438
 symmetric_difference() (метод axipy.RoundRectangle), 450
 symmetric_difference() (метод axipy.Text), 488
 SystemActionButton (класс в ахипу), 696

T

tab (axipy.ProviderManager property), 204
 TabDataProvider (класс в ахипу), 222
 TabFile (класс в ахипу), 540
 Table (класс в ахипу), 267
 table (axipy.SelectionManager property), 676
 table() (метод axipy.TableReportItem), 627
 table_data_changed (axipy.DataManager property), 263
 table_view (axipy.TableView property), 651
 TableReportItem (класс в ахипу), 626
 tables (axipy.DataManager property), 264
 TableView (класс в ахипу), 649
 tableviews (axipy.ViewManager property), 680
 Task (класс в ахипу), 167
 task_manager (в модуле ахипу), 142
 Task.CanceledException, 169
 TaskManager (класс в ахипу), 179
 Task.Range (класс в ахипу), 169
 Task.Status (класс в ахипу), 170
 Text (класс в ахипу), 475
 text (axipy.CollectionStyle property), 531
 text (axipy.Label property), 573
 text (axipy.Text property), 489
 text_style (axipy.CustomLabelProperties property), 576
 TextAlignment (класс в ахипу), 527
 TextBackgroundType (класс в ахипу), 527
 TextCallout (класс в ахипу), 527
 TextStyle (класс в ахипу), 522
 thematic (axipy.CosmeticLayer property), 568
 thematic (axipy.VectorLayer property), 564
 ThematicLayer (класс в ахипу), 580
 thread_id (axipy.DialogTask property), 179
 thread_id (axipy.Task property), 172
 time() (статический метод axipy.Attribute), 311
 title (axipy.BarThematicLayer property), 600
 title (axipy.CoordSystem property), 158
 title (axipy.CosmeticLayer property), 568
 title (axipy.DensityThematicLayer property), 613
 title (axipy.DialogTask property), 179
 title (axipy.DrawableView property), 655
 title (axipy.IndividualThematicLayer property), 609
 title (axipy.Layer property), 556
 title (axipy.LegendItem property), 579
 title (axipy.LegendView property), 672
 title (axipy.ListLayers property), 552
 title (axipy.MapView property), 663
 title (axipy.PieThematicLayer property), 594
 title (axipy.RangeThematicLayer property), 588
 title (axipy.RasterLayer property), 559
 title (axipy.ReportView property), 669
 title (axipy.SymbolThematicLayer property), 603
 title (axipy.TableView property), 651
 title (axipy.Task property), 172
 title (axipy.VectorLayer property), 564
 title (axipy.View property), 649
 title_changed (axipy.DialogTask property), 179
 title_changed (axipy.Task property), 172
 tms (axipy.ProviderManager property), 204
 TmsDataProvider (класс в ахипу), 238
 to_device() (метод axipy.ViewTool), 640
 to_geojson() (метод axipy.Arc), 474
 to_geojson() (метод axipy.Ellipse), 462
 to_geojson() (метод axipy.Feature), 304
 to_geojson() (метод axipy.Geometry), 324
 to_geojson() (метод axipy.GeometryCollection), 388
 to_geojson() (метод axipy.Line), 348
 to_geojson() (метод axipy.LineString), 360
 to_geojson() (метод axipy.MultiLineString), 414
 to_geojson() (метод axipy.MultiPoint), 401
 to_geojson() (метод axipy.MultiPolygon), 427
 to_geojson() (метод axipy.Point), 336
 to_geojson() (метод axipy.Polygon), 373
 to_geojson() (метод axipy.Rectangle), 439
 to_geojson() (метод axipy.RoundRectangle), 450
 to_geojson() (метод axipy.Text), 489
 to_image() (метод axipy.Legend), 578
 to_image() (метод axipy.Map), 549
 to_linestring() (метод axipy.Arc), 474
 to_linestring() (метод axipy.Ellipse), 462

to_linestring() (метод axipy.Geometry), 324
 to_linestring() (метод axipy.GeometryCollection), 388
 to_linestring() (метод axipy.Line), 348
 to_linestring() (метод axipy.LineString), 360
 to_linestring() (метод axipy.MultiLineString), 414
 to_linestring() (метод axipy.MultiPoint), 401
 to_linestring() (метод axipy.MultiPolygon), 427
 to_linestring() (метод axipy.Point), 336
 to_linestring() (метод axipy.Polygon), 373
 to_linestring() (метод axipy.Rectangle), 439
 to_linestring() (метод axipy.RoundRectangle), 450
 to_linestring() (метод axipy.Text), 489
 to_mapinfo() (метод axipy.CollectionStyle), 531
 to_mapinfo() (метод axipy.FillStyle), 518
 to_mapinfo() (метод axipy.LineStyle), 515
 to_mapinfo() (метод axipy.PointCompatStyle), 503
 to_mapinfo() (метод axipy.PointFontStyle), 508
 to_mapinfo() (метод axipy.PointPictureStyle), 512
 to_mapinfo() (метод axipy.PointStyle), 499
 to_mapinfo() (метод axipy.PolygonStyle), 522
 to_mapinfo() (метод axipy.Style), 495
 to_mapinfo() (метод axipy.TextStyle), 526
 to_mif() (метод axipy.Arc), 474
 to_mif() (метод axipy.Ellipse), 463
 to_mif() (метод axipy.Geometry), 324
 to_mif() (метод axipy.GeometryCollection), 388
 to_mif() (метод axipy.Line), 348
 to_mif() (метод axipy.LineString), 360
 to_mif() (метод axipy.MultiLineString), 414
 to_mif() (метод axipy.MultiPoint), 401
 to_mif() (метод axipy.MultiPolygon), 427
 to_mif() (метод axipy.Point), 336
 to_mif() (метод axipy.Polygon), 373
 to_mif() (метод axipy.Rectangle), 439
 to_mif() (метод axipy.RoundRectangle), 450
 to_mif() (метод axipy.Text), 489
 to_normalized() (метод axipy.AngleCoord), 195
 to_polygon() (метод axipy.Arc), 475
 to_polygon() (метод axipy.Ellipse), 463
 to_polygon() (метод axipy.Geometry), 324
 to_polygon() (метод axipy.GeometryCollection), 388
 to_polygon() (метод axipy.Line), 348
 to_polygon() (метод axipy.LineString), 360
 to_polygon() (метод axipy.MultiLineString), 414
 to_polygon() (метод axipy.MultiPoint), 401
 to_polygon() (метод axipy.MultiPolygon), 427
 to_polygon() (метод axipy.Point), 336
 to_polygon() (метод axipy.Polygon), 373
 to_polygon() (метод axipy.Rectangle), 439
 to_polygon() (метод axipy.RoundRectangle), 451
 to_polygon() (метод axipy.Text), 489
 to_qt() (метод axipy.Pnt), 185
 to_qt() (метод axipy.Rect), 190
 to_scene() (метод axipy.ViewTool), 641
 to_string() (метод axipy.CoordSystem), 158
 to_tuple() (метод axipy.Pnt), 185
 to_unit() (метод axipy.AreaUnit), 164
 to_unit() (метод axipy.LinearUnit), 162
 to_wkb() (метод axipy.Arc), 475
 to_wkb() (метод axipy.Ellipse), 463
 to_wkb() (метод axipy.Geometry), 324
 to_wkb() (метод axipy.GeometryCollection), 388
 to_wkb() (метод axipy.Line), 348
 to_wkb() (метод axipy.LineString), 361
 to_wkb() (метод axipy.MultiLineString), 414
 to_wkb() (метод axipy.MultiPoint), 401
 to_wkb() (метод axipy.MultiPolygon), 427
 to_wkb() (метод axipy.Point), 336
 to_wkb() (метод axipy.Polygon), 373
 to_wkb() (метод axipy.Rectangle), 439
 to_wkb() (метод axipy.RoundRectangle), 451
 to_wkb() (метод axipy.Text), 489
 ToolButton (класс в axipy), 697
 touches() (метод axipy.Arc), 475
 touches() (метод axipy.Ellipse), 463
 touches() (метод axipy.Geometry), 324
 touches() (метод axipy.GeometryCollection), 388
 touches() (метод axipy.Line), 348
 touches() (метод axipy.LineString), 361
 touches() (метод axipy.MultiLineString), 414
 touches() (метод axipy.MultiPoint), 401
 touches() (метод axipy.MultiPolygon), 427
 touches() (метод axipy.Point), 336
 touches() (метод axipy.Polygon), 373
 touches() (метод axipy.Rectangle), 439
 touches() (метод axipy.RoundRectangle), 451
 touches() (метод axipy.Text), 489
 tr() (метод axipy.Plugin), 132
 transform() (в модуле axipy), 535
 transform() (метод axipy.CoordTransformer), 160
 translated() (метод axipy.Rect), 190
 transparentColor (axipy.RasterLayer property), 559
 try_enable() (метод axipy.AxipyAcceptableActiveToolHandler), 646
 try_to_simplified() (метод axipy.GeometryCollection), 389
 try_to_simplified() (метод axipy.MultiLineString), 414
 try_to_simplified() (метод axipy.MultiPoint), 402
 try_to_simplified() (метод axipy.MultiPolygon), 427
 type (axipy.Arc property), 475
 type (axipy.Ellipse property), 463
 type (axipy.Geometry property), 324
 type (axipy.GeometryCollection property), 389
 type (axipy.Line property), 348
 type (axipy.LineString property), 361
 type (axipy.MultiLineString property), 414
 type (axipy.MultiPoint property), 402
 type (axipy.MultiPolygon property), 427
 type (axipy.Point property), 336
 type (axipy.Polygon property), 374
 type (axipy.Rectangle property), 439
 type (axipy.RoundRectangle property), 451
 type (axipy.Text property), 489
 type_string (axipy.Attribute property), 311
 typedef (axipy.Attribute property), 311
 TypeSqlDialect (класс в axipy), 531

U

underline (axipy.TextStyle property), 527

undo() (метод axipy.CosmeticTable), 300
 undo() (метод axipy.DrawableView), 655
 undo() (метод axipy.MapView), 663
 undo() (метод axipy.QueryTable), 283
 undo() (метод axipy.ReportView), 669
 undo() (метод axipy.SelectionTable), 292
 undo() (метод axipy.Table), 275
 ungroup() (метод axipy.ListLayers), 552
 union() (метод axipy.Arc), 475
 union() (метод axipy.Ellipse), 463
 union() (метод axipy.Geometry), 325
 union() (метод axipy.GeometryCollection), 389
 union() (метод axipy.Line), 348
 union() (метод axipy.LineString), 361
 union() (метод axipy.MultiLineString), 414
 union() (метод axipy.MultiPoint), 402
 union() (метод axipy.MultiPolygon), 427
 union() (метод axipy.Point), 336
 union() (метод axipy.Polygon), 374
 union() (метод axipy.Rectangle), 439
 union() (метод axipy.RoundRectangle), 451
 union() (метод axipy.Text), 489
 unique (axipy.Attribute property), 311
 unit (axipy.BoundingRectDialog property), 686
 unit (axipy.CoordSystem property), 158
 unit (axipy.MapView property), 663
 unit (axipy.Report property), 618
 unit (axipy.UnitValue property), 166
 UnitValue (класс в ахипу), 165
 unload() (метод axipy.Plugin), 132
 unload() (метод axipy.PluginManager), 133
 unload() (метод axipy.ViewTool), 641
 unloaded (axipy.PluginManager property), 133
 UP (атрибут axipy.BarThematicLayer), 598
 UP (атрибут axipy.OrientationThematic), 615
 UP (атрибут axipy.PieThematicLayer), 591
 update() (метод axipy.CosmeticTable), 300
 update() (метод axipy.GeometryReportItem), 622
 update() (метод axipy.LegendReportItem), 629
 update() (метод axipy.MapReportItem), 624
 update() (метод axipy.QueryTable), 283
 update() (метод axipy.RasterReportItem), 626
 update() (метод axipy.ReportItem), 620
 update() (метод axipy.ScaleBarReportItem), 630
 update() (метод axipy.SelectionTable), 292
 update() (метод axipy.Table), 275
 update() (метод axipy.TableReportItem), 627
 updated (axipy.DataManager property), 264
 UseAntialiasing (атрибут axipy.CurrentSettings), 138
 useClip (axipy.Label property), 573
 UseLastSelectedFilter (атрибут axipy.CurrentSettings), 138
 UseNativeFileDialog (атрибут axipy.CurrentSettings), 138
 user_name (axipy.PasswordDialog property), 685
 UserDataFolder (атрибут axipy.CurrentSettings), 138
 UserDataPaths (атрибут axipy.CurrentSettings), 139
 UserPluginsDataFolder (атрибут axipy.CurrentSettings), 139
 UserPluginsDependenciesFolder (атрибут axipy.CurrentSettings), 139
 UserPluginsFolder (атрибут axipy.CurrentSettings), 139
 UserPluginsInstallationFolder (атрибут axipy.CurrentSettings), 139
 UserPluginsSettingsFolder (атрибут axipy.CurrentSettings), 139

V

value (axipy.AngleCoord property), 196
 value (axipy.DialogTask property), 179
 value (axipy.FloatCoord property), 193
 value (axipy.Observer property), 539
 value (axipy.Task property), 172
 value (axipy.UnitValue property), 166
 value_changed (axipy.DialogTask property), 179
 value_changed (axipy.Task property), 172
 values() (метод класса axipy.CurrentSettings), 139
 values() (метод axipy.ActionManager), 682
 values() (метод axipy.Feature), 305
 values() (метод axipy.ObserverManager), 538
 values() (метод axipy.TaskManager), 182
 VectorLayer (класс в ахипу), 560
 Version (класс в ахипу), 150
 vertical_pages (axipy.Report property), 618
 View (класс в ахипу), 647
 view (axipy.ViewTool property), 641
 view_changed (axipy.ViewManagerWidget property), 691
 view_manager (в модуле ахипу), 142
 view_scale (axipy.ReportView property), 669
 ViewManager (класс в ахипу), 676
 ViewManagerWidget (класс в ахипу), 690
 views (axipy.ViewManager property), 680
 ViewTool (класс в ахипу), 635
 ViewTool.ViewType (класс в ахипу), 637
 visible (axipy.BarThematicLayer property), 600
 visible (axipy.CosmeticLayer property), 568
 visible (axipy.DensityThematicLayer property), 613
 visible (axipy.IndividualThematicLayer property), 609
 visible (axipy.Label property), 573
 visible (axipy.LabelLayout property), 574
 visible (axipy.Layer property), 556
 visible (axipy.LegendItem property), 579
 visible (axipy.ListLayers property), 552
 visible (axipy.PieThematicLayer property), 594
 visible (axipy.RangeThematicLayer property), 588
 visible (axipy.RasterLayer property), 559
 visible (axipy.SymbolThematicLayer property), 603
 visible (axipy.VectorLayer property), 564

W

Warning (атрибут axipy.Notifications), 692
 WebOpenData (класс в ахипу), 246
 wheelEvent() (метод axipy.ViewTool), 641
 white_border (axipy.PointFontStyle property), 508
 widget (axipy.AxipyActiveToolPanelHandlerBase property), 645
 widget (axipy.DataManagerWidget property), 690
 widget (axipy.DrawableView property), 655
 widget (axipy.LayerControlWidget property), 689
 widget (axipy.LegendView property), 672
 widget (axipy.MainWindow property), 150
 widget (axipy.MapView property), 663
 widget (axipy.NotificationWidget property), 691
 widget (axipy.PythonConsoleWidget property), 691
 widget (axipy.ReportView property), 669
 widget (axipy.TableView property), 651
 widget (axipy.View property), 649
 widget (axipy.ViewManagerWidget property), 691
 width (axipy.LineStyle property), 515
 width (axipy.Rect property), 190
 width (axipy.Text property), 489
 within() (метод axipy.Arc), 475
 within() (метод axipy.Ellipse), 463

`within()` (метод `axipy.Geometry`), 325
`within()` (метод `axipy.GeometryCollection`), 389
`within()` (метод `axipy.Line`), 348
`within()` (метод `axipy.LineString`), 361
`within()` (метод `axipy.MultiLineString`), 415
`within()` (метод `axipy.MultiPoint`), 402
`within()` (метод `axipy.MultiPolygon`), 427
`within()` (метод `axipy.Point`), 336
`within()` (метод `axipy.Polygon`), 374
`within()` (метод `axipy.Rectangle`), 439
`within()` (метод `axipy.RoundRectangle`), 451
`within()` (метод `axipy.Text`), 489
`wkt` (`axipy.CoordSystem` property), 158
`wms` (`axipy.ProviderManager` property), 204
`WmsDataProvider` (класс в `axipy`), 244
`wmts` (`axipy.ProviderManager` property), 204
`WmtsDataProvider` (класс в `axipy`), 246
`Workspace` (класс в `axipy`), 682

X

`x` (`axipy.Pnt` property), 185
`x` (`axipy.Point` property), 336
`x_guidelines` (`axipy.ReportView` property), 669
`xmax` (`axipy.Rect` property), 190
`xmax` (`axipy.Rectangle` property), 439
`xmax` (`axipy.RoundRectangle` property), 451
`xmin` (`axipy.Rect` property), 190
`xmin` (`axipy.Rectangle` property), 439
`xmin` (`axipy.RoundRectangle` property), 451
`xRadius` (`axipy.Arc` property), 475
`xRadius` (`axipy.RoundRectangle` property), 451

Y

`y` (`axipy.Pnt` property), 185
`y` (`axipy.Point` property), 337
`y_guidelines` (`axipy.ReportView` property), 669
`ymax` (`axipy.Rect` property), 190
`ymax` (`axipy.Rectangle` property), 439
`ymax` (`axipy.RoundRectangle` property), 451
`ymin` (`axipy.Rect` property), 190
`ymin` (`axipy.Rectangle` property), 439
`ymin` (`axipy.RoundRectangle` property), 451
`yRadius` (`axipy.Arc` property), 475
`yRadius` (`axipy.RoundRectangle` property), 451

Z

`zoom()` (метод `axipy.MapView`), 663
`zoom_restrict` (`axipy.BarThematicLayer` property), 600
`zoom_restrict` (`axipy.CosmeticLayer` property), 568
`zoom_restrict` (`axipy.DensityThematicLayer` property), 613
`zoom_restrict` (`axipy.IndividualThematicLayer` property), 609
`zoom_restrict` (`axipy.Layer` property), 556
`zoom_restrict` (`axipy.PieThematicLayer` property), 594
`zoom_restrict` (`axipy.RangeThematicLayer` property), 588
`zoom_restrict` (`axipy.RasterLayer` property), 559
`zoom_restrict` (`axipy.SymbolThematicLayer` property), 603
`zoom_restrict` (`axipy.VectorLayer` property), 564