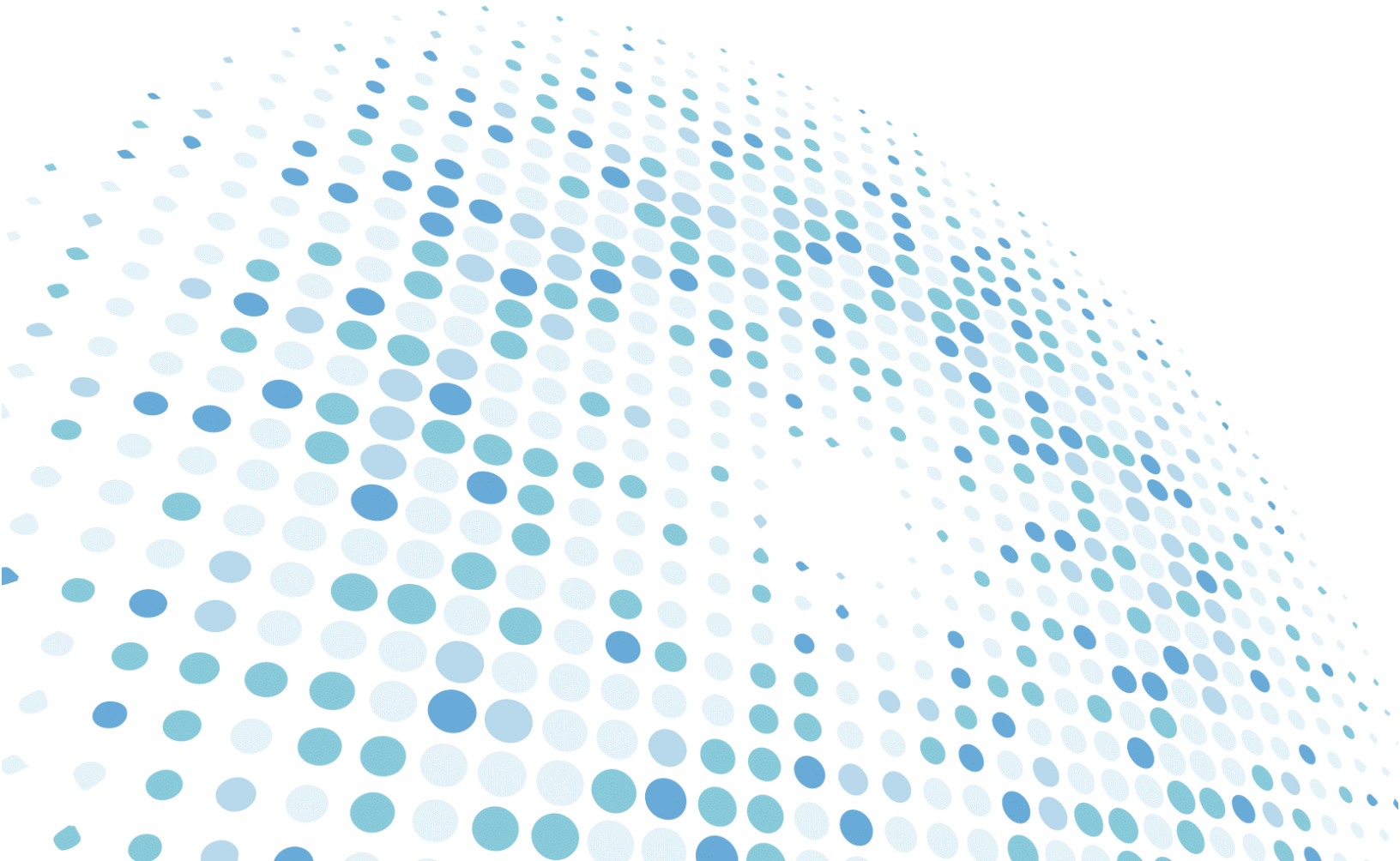
Цифровая платформа
«Управление в пространстве» (ЦП УВП)

# ОПИСАНИЕ МЕТОДОВ ЯДРА



# Содержание




© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Описание функциональности

## Основные возможности

Блок Python позволяет взаимодействовать с функциями ядра самой платформы ЦП УвП. Наборы классов, которые доступны разработчику, предоставляют возможность управления как внутренними состояниями и методами объектов, так и самими объектами.

Перечень объектов, с которыми возможно взаимодействие:

1. Справочники (данные справочников) – класс CDictionary


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionary

Блок Python позволяет взаимодействовать с новой структурой справочников. Для инициализации класса CDictionary, отвечающего за взаимодействие с данными справочника, необходимо вызвать метод `**execution_context.create_dictionary_instance()**`, с параметрами:

- **dictionary_id**: идентификатор справочника.
- **revision**: идентификатор ревизии данных.
- **blockid**: блок-источник данных (параметр _source), не обязательное.

После завершения работы со справочником необходимо вызвать метод .clear()

**Базовые классы**

*CDictionary*

Основной класс для работы с сервисом справочника. Создает окружение и отправляет запросы на шину или API для создания, расчета, работы с данными справочника.

Доступные параметры:

- - **blockid**: Optional[str]
  - Идентификатор блока, в котором создан справочник
- - **user_id**: Optional[str]
  - Идентификатор пользователя, для которого создан справочник
- - **dictionary_type**: EDictionaryTypes
  - Тип справочника
- - **attributes**: CAttributes
  - Список атрибутов справочника
- - **binding**: Dict
  - Словарь с привязками полей справочника
- - **lazy_load**: bool
  - Флаг ленивой (отложенной) загрузки данных справочника
- - **dictionary_id**: Optional[str]
  - Идентификатор справочника
- - **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника
- - **specific_params:** Dict
  - Специфические параметры для инициализации справочника
- - **hierarchy**: Optional[CDictionaryHierarchyManager]
  - Иерархии справочника

Доступные методы:


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- - **clear**() -> None:
    - Очистка справочника, для последующего корректного удаления сборщиком мусора
- - **send_to_bus**(action: str) -> None:
    - Передача сообщения на шину и получение ответа.
        - - action: действие со словарем
- - **get_data**(limit: Optional[int], offset: Optional[int], get_by_block: bool) -> CDictionaryElements
    - Получение всех рассчитанных данных справочника. В случае наличия иерархий - иерархия по умолчанию будет применена к данным.
        - - limit: Количество объектов
        - - offset: Сдвиг объектов
        - - get_by_block: Флаг поиска справочника
        - по блоку
- - **get_origin_data**(limit: Optional[int], offset: Optional[int], get_by_block: bool) -> CDictionaryElements
    - Получение всех рассчитанных данных справочника.  Данные приходят в зависимости от наличия иерархий в справочнике.
        - - limit: Количество объектов
        - - offset: Сдвиг объектов
        - - get_by_block: Флаг поиска справочника по блоку
- - **invalidate_cache**(by_block: bool) -> None:
    - Инвалидация кеша справочника.
        - - by_block: флаг инвалидации всех справочников, привязанных к блоку .Инвалидация проходит по параметру block_id, а не по dictionary_id
- - **get_last_delta**() -> Optional[str]
    - Получение идентификатора ревизии для последней (по времени создания) дельты справочника
- - **get_all_deltas**() -> List[Dict]
    - Получение списка всех дельт для справочника
- - **get_revision_history**() -> List[Dict]
    - Получение истории ревизий справочника, от момента создания и до ревизии, указанной в revision_id
- - **save_delta**(delta: Dict) -> Optional[str]
    - Сохранение дельты справочника и получение нового идентификатора ревизии. Новая ревизия становится дочерней к той, которая указана в revision_id.
        - - delta: Словарь с изменениями данных, формата {<insert>, <update>, <delete>}.
        - <insert>, <update>, <delete> являются массивами, содержащими словари формата
        - {"keys": {<id>: <value>}, "value": {<id>: <value>}}


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- - `keys` - словарь ключей (атрибутов, у которых параметр primary_key=1);
- - `value` - словарь всех атрибутов (в том числе и ключей);
- - `<id>` - идентификатор атрибута;
- - `<value>` - значение атрибута.

*CDictionaryElements*

Класс по работе с массивом элементов справочника. Приходит как результат любого метода по получению данных.

Доступные параметры:

- - **block_id**: Optional[str]
  - Идентификатор блока справочника, из которого получены данные
- - **dict_id**: Optional[str]
  - Идентификатор справочника, из которого получены данные
- - **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника, из которого получены данные
- - **binding**: Dict
  - Словарь с привязками данных справочника
- - **attributes**: CAttributes
  - Метаданные всех атрибутов справочника

Доступные свойства:

- - **count** -> int
  - Общее количество элементов в справочнике

Доступные методы:

- - **clear**() -> None
  - Очистка всех элементов справочника
- - **fill**(raw_data: Dict) -> None
  - Заполнение (с перезаписью) справочника данными и метаданными (binding, attributes)
    - - raw_data: словарь с необработанными данными в формате {<fields>, <binding>, <data>}
- - **fill_data**(raw_data: List[Dict], clear: bool) -> None
  - Заполнение справочника данными
    - - raw_data: массив с данными в формате {<key>: <value>}
    - - clear: флаг очистки старых данных.
- - **get_fields**(reverse_not_null: bool) -> CDataFields


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

    - Получение списка полей в формате CDataFields. Устарело, используйте .attributes.serialize()
        - - reverse_not_null: флаг инвертирования параметра not_null
- - **get_elements**(flatten: bool) -> List[Union[CDictionaryElement, CDictionaryHierarchyElement]]
    - Получение массива элементов справочника.
    - - flatten: отображение иерархических данных в виде плоского списка.
- - **to_json**() -> List[Dict]
    - Получение плоского массива с элементами справочника в формате {<key>: <value>}
- - **to_old_format_json**() -> List[Dict]
    - Получение массива элементов справочника в устаревшем формате объектов. Устарело, используйте to_json() или serialize()
- - **serialize**() -> Dict
    - Сериализация данных и метаданных справочника в формат {<fields>, <binding>, <data>}
- - **get_element_by_primary_keys**(keys: Dict) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]
    - Получение элемента по первичным ключам
        - - keys: словарь формата {<key>: <value>}
- - **get_element_from_alt_hierarchy_element**(alt_hierarchy_element: Union[CDictionaryElement, CDictionaryHierarchyElement]) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]
    - Получение оригинального элемента по элементу из альтернативной иерархии.
        - - alt_hierarchy_element: элемент из альтернативной иерархии.
- - **get_element_from_alt_hierarchy_element_old_format**(alt_hierarchy_element: Dict) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]:
    - Получение оригинального элемента по элементу из альтернативной иерархии. Элемент приходит в устаревшем формате
        - - alt_hierarchy_element: элемент из альтернативной иерархии (в формате словаря).

**Работа с иерархиями**

*CDictionaryHierarchyManager*

Менеджер по работе с иерархиями. Подключается при инициализации CDictionary.

Заполнение происходит при вызове метода .fill_from_attributes([CAttributes]). Анализируются все атрибуты справочника, и на их основе формируются иерархии (т.к. каждой иерархии соответствует одно либо несколько полей)

Доступные параметры:


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- - **dictionary_id**: str
    - Идентификатор справочника, к которому подключен менеджер
- - **user_id**: Optional[str]
    - Идентификатор пользователя справочника, к которому подключен менеджер
- - **block_id**: Optional[str]
    - Идентификатор блока справочника, к которому подключен менеджер
- - **revision_id**: Optional[str]
    - Идентификатор ревизии данных справочника, к которому подключен менеджер

Доступные свойства:

- - **count** -> [int]
    - Количество иерархий
- - **items** -> List[Dict]
    - Список всех иерархий в формате {<name>, <type>, <default>}
- - **default** -> CDictionaryHierarchy
    - Получение иерархии по умолчанию (либо первой в списке, если по умолчанию не было отмечено иерархий)
- - **basic** -> Optional[CDictionaryBasicHierarchy]
    - Получение базовой иерархии (родитель - ребенок), если она есть
- - **enabled** -> bool
    - Флаг наличия иерархий

Доступные методы:

- - **clear**() -> None
    - Очистка всех иерархий
- - **fill_from_attributes**(attributes: CAttributes) -> None
    - Заполнение иерархий исходя из метаданных атрибутов справочника

*CDictionaryHierarchy*

Абстрактный класс для всех типов иерархий в справочниках

После окончания работы необходимо очистить методом .clear()

Доступные параметры:

- - **name**: str
    - Наименование иерархии
- - **basic**: bool
    - Флаг того, что иерархия является базовой (родитель - ребенок)
- - **default**: bool


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

    - Флаг иерархии по умолчанию
- - **dictionary_id**: str
    - Идентификатор справочника, для которого строится иерархия
- - **user_id**: Optional[str]
    - Идентификатор пользователя справочника, для которого строится иерархия
- - **block_id**: Optional[str]
    - Идентификатор блока справочника, для которого строится иерархия
- - **revision_id**: Optional[str]
    - Идентификатор ревизии данных справочника, для которого строится иерархия

Доступные методы:

- - **clear**() -> None
    - Очистка всех элементов класса
- - **get_data**(limit: Optional[int], offset: Optional[int], get_by_block: bool) -> CDictionaryElements
    - Получение всех данных справочника
        - - limit: Количество объектов
        - - offset: Сдвиг объектов
        - - get_by_block: Флаг поиска справочника по блоку
- - **get_child_elements**(parent: Union[CDictionaryHierarchyElement, str], all_child: bool) -> CDictionaryElements
    - Получение дочерних элементов (одного или всего дерева) для конкретного родителя
    - - parent: родительский элемент
    - - all_child: флаг для получения всего дерева дочерних элементов
- - **get_parent_element**(parent: Union[CDictionaryHierarchyElement, str]) -> CDictionaryElements
    - Получение родителя выбранного элемента
        - - parent: родительский элемент
- - **get_elements_on_level**(level: int) -> CDictionaryElements
    - Получение элементов по одному уровню
        - - level: уровень справочника

*CDictionaryBasicHierarchy*

Класс для получения базовой иерархии (родитель-ребенок).

После окончания работы необходимо очистить методом .clear()

Доступные параметры:

Описание доступных параметров см. в CDictionaryHierarchy

Доступные методы:


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

Описание доступных методов см. в CDictionaryHierarchy

*CDictionaryAlternativeHierarchy*

Класс для получения альтернативных иерархий, справочников, связанных с оригинальным по ключам определенного поля.

После окончания работы необходимо очистить методом .clear()

Доступные параметры:

Описание доступных параметров см. в CDictionaryHierarchy. Плюс к ним добавляются:

- - **key**: str
    - Ключ атрибута, отвечающего за иерархию
- - **alt_block**: str
    - Идентификатор блока с альтернативной иерархией
- - **alt_block_key**: str
    - Ключ атрибута в блоке с альтернативной иерархией для связи ключей
- - **alt_revision**: str
    - Ревизия данных блока, для которой отображаем иерархию

Доступные методы:

Описание доступных методов см. в CDictionaryHierarchy.

**Работа с единичными элементами**

*CAttributes*

Класс для взаимодействия с массивом атрибутов. Учитывается только мета информация по атрибутам (имя, тип, и прочие), значения атрибута не учитываются.

Доступные параметры:

- - **attributes**: Dict[str, CAttribute]
    - Словарь с атрибутами формата {<attribute_id>: <attribute>}

Доступные свойства:

- - **empty** -> bool
    - Флаг отсутствия элементов атрибута. Не учитывает скрытые атрибуты
- - **all_id** -> List[str]
    - Получение идентификаторов всех атрибутов. Не учитывает скрытые атрибуты
- - **primary_keys** -> List[CAttribute]
    - Список первичных ключей


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- - **primary_keys_id** -> List[str]
    - Список идентификаторов первичных ключей
- - **has_hidden_attributes** -> bool
    - Флаг наличия скрытых атрибутов
- - **hidden_attributes** -> List[CAttribute]
    - Получение всех скрытых атрибутов
- - **has_basic_hierarchy** -> bool
    - Флаг наличия базовой иерархии
- - **has_alt_hierarchy** -> bool
    - Флаг наличия атрибутов, отвечающих за альтернативную иерархию
- - **alt_hierarchy_attributes** -> List[CAttribute]
    - Список атрибутов, отвечающих за альтернативную иерархию
- - **structure** -> Dict
    - Получение структуры объекта в формате {<name>, <id>, <type>, <mandatory (not null)>, <primary_key>}

Доступные методы:

- - **__getitem__**(field_id: str) -> Optional[CAttribute]
    - Получение атрибута по его идентификатору (поле 'id'). Переопределение встроенного метода, с форматом `CAttributes[<field_id>]`
        - - field_id: идентификатор атрибута
- - **clear**() -> None
    - Очистка атрибутов.
- - **update_attributes**(fields: List[Dict]) -> None
    - Обновление списка атрибутов. Старые атрибуты будут затерты.
        - - fields: массив с атрибутами
- - **serialize**() -> List[Dict]
    - Сериализация атрибутов.

*CDictionaryElement*

Класс для взаимодействия с массивом объектов справочника. Построен на основе CAttributes, но тут так же учитываются значения элементов.

Доступные параметры:

Описание доступных параметров см. в CAttributes

Доступные свойства:

Описание доступных свойств см. в CAttributes. Плюс к ним добавляются:

- - **hash_primary_key** -> int
    - Хеш значение первичного ключа (учитывает значение атрибутов, входящих в первичный ключ)


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

Доступные методы:

Описание доступных методов см. в CAttributes. Плюс к ним добавляются:

- - **fill_from_attributes**(attributes: CAttributes, elements: Optional[Dict]) -> None
    - Заполнение элементов с использованием уже созданного класса CAttributes.
        - - attributes: класс CAttributes
        - - elements: словарь со значениями атрибутов в формате {<id>: <value>}
- - **to_json**() -> Dict
    - Сериализация данных справочника в формат {<id>: <value>}
- - **to_old_format_json**(binding: Dict, block_id: str, dict_id: Optional[str], revision_id: Optional[str]) -> Dict
    - Получение массива элементов справочника в устаревшем формате объектов. Устарело, используйте to_json() или serialize()
        - - binding: параметры привязки блока.
        - - block_id: идентификатор блока.
        - - dict_id: идентификатор справочника на сервисе.
        - - revision_id: идентификатор ревизии данных.
- - **serialize**() -> Dict
    - Сериализация атрибутов и данных справочника в словарь вида {<elements>, <fields>}. Где данные справочника (<elements>) сериализуются методом to_json(), а атрибуты (<fields>) CAttributes.serialize()

*CDictionaryHierarchyElement*

Класс для взаимодействия с массивом объектов справочника. Аналогичен CDictionaryElement, но тут добавляются параметры для работы с иерархическими объектами

Доступные параметры:

Описание доступных параметров см. в CDictionaryElement. Плюс к ним добавляются:

- - **child**: List[CDictionaryHierarchyElement]
    - Массив дочерних элементов объекта
- - **level**: int
    - Уровень объекта в иерархии. Начинается с нуля

Доступные свойства:

Описание доступных свойств см. в CDictionaryElement. Плюс к ним добавляются:

- - **parent_key** -> CAttribute
    - Получение атрибута, отвечающего за родителя в базовой иерархии


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- - **hierarchy_key** -> CAttribute
    - Получение атрибута, отвечающего за уникальный ключ для связи базовой иерархии

Доступные методы:

Описание доступных методов см. в CDictionaryElement. Плюс к ним добавляются:

- - **clear**() -> None
    - Очистка объектов справочника.
- - **to_json**() -> Dict
    - Сериализация данных справочника в формате массива [{<id>: <value>},...]
- - **to_old_format_json**(binding: Dict, block_id: str, dict_id: Optional[str], revision_id: Optional[str]) -> Dict
    - Получение массива элементов справочника в устаревшем формате объектов. Отличие от родительского (CDictionaryElement) метода, здесь собирается иерархия и добавляется параметр _hierarchy_level. Устарело, используйте to_json() или serialize()
        - - binding: параметры привязки блока.
        - - block_id: идентификатор блока.
        - - dict_id: идентификатор справочника на сервисе.
        - - revision_id: идентификатор ревизии данных.
- - **serialize**() -> Dict
    - Сериализация атрибутов и данных справочника в словарь вида {<elements>, <fields>}. Где данные справочника (<elements>) сериализуются методом to_json(), а атрибуты (<fields>) CAttributes.serialize()


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CAttributes

Класс для взаимодействия с атрибутами объектов.

Учитывается только метаинформация по атрибутам (имя, тип), значения не учитываются.

После завершения работы необходимо вызвать метод .clear()

**Импорт**

```
from src.service_dictionary.elements.attributes import CAttributes
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он приходит при получении атрибутов справочника

```
CAttributes(
   fields: Optional[List[Dict]] = None
)
```

- **fields** - массив с описанием атрибутов.

Можно не передавать атрибуты при инициализации, а добавить их позднее, методами **add_attributes** или **refresh_attributes**.

**Параметры**

- **attributes**: Dict[str, CAttribute]
    - Словарь с атрибутами формата {<attribute_id>: <attribute>}

**Свойства**

- **empty** -> bool
    - Флаг отсутствия элементов атрибута. Не учитывает системные атрибуты
- **has_system_attributes** -> bool
    - Флаг наличия системных атрибутов
- **system_attributes** -> List[CAttribute]
    - Получение всех системных атрибутов
- **has_basic_hierarchy** -> bool
    - Флаг наличия базовой иерархии
- **has_alt_hierarchy** -> bool
    - Флаг наличия атрибутов, отвечающих за альтернативную иерархию
- **alt_hierarchy_attributes** -> List[CAttribute]
    - Список атрибутов, отвечающих за альтернативную иерархию
- **structure** -> Dict
    - Получение структуры объекта в формате


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

{<name>, <id>, <type>, <mandatory (not null)>, <primary_key>}

**Методы**

Сериализация

```
serialize() -> List[Dict]
```

Сериализация атрибутов справочника массив

Очистка класса

```
clear() -> None
```

Очистка класса справочника, для его корректного сборщиком мусора.

Получение атрибута по идентификатору

```
__getitem__(key: str) -> Optional[CAttribute]
```

```
attributes = CAttributes(...)
attribute = attributes[key]
```

Получение атрибута по его ключу.

В случае отсутствия такого ключа атрибута - вернется None.

- **key** - ключ атрибута

Изменение атрибута по идентификатору

```
__setitem__(key: str, value: CAttribute) -> None:
```

```
attributes = CAttributes(...)
attributes[key] = new_attribute
```

Изменение атрибута по его ключу.

- **key** - ключ атрибута
- **value** - атрибут

Получение количетсва атрибутов

```
__len__() -> int
```

```
attributes = CAttributes(...)
count = len(attributes)
```

Получение количества атрибутов.


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

Получение первичных ключей

```
primary_keys(system: bool = False) -> List[CAttribute]
```

Получение идентификаторов первичных ключей

```
primary_keys_id(system: bool = False) -> List[str]
```

Проверка наличия атрибута

```
has_selected_id(ident: str) -> bool
```

Получение атрибута по идентификатору

```
get_by_id(ident: str) -> Optional[CAttribute]
```

Получение идентификаторов всех атрибутов

```
all_id(self, system: bool = False) -> List[str]
```

Обновление списка атрибутов

```
refresh_attributes(self, fields: List[Dict] = ()) -> None
```

Добавление новых атрибутов

```
add_attributes(self, fields: List[Dict] = ()) -> None
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryElements

Класс по работе с элементами справочника с данными.

Приходит как результат любого метода по получению данных.

После завершения работы со справочником необходимо вызвать метод .clear()

**Импорт**

```
from src.service_dictionary.c_dictionary_elements import CDictionaryElements
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он приходит при получении данных справочника.

```
CDictionaryElements(
   block_id: Optional[str] = None,
   dict_id: Optional[str] = None,
   revision_id: Optional[str] = None,
   binding: Dict = {},
   attributes: CAttributes = CAttributes()
)
```

- **block_id** - идентификатор блока справочника.
- **dict_id** - идентификатор справочника.
- **revision_id** - идентификатор ревизии данных справочника.
- **binding** - описание привязок справочника.
- **attributes** - описание атрибутов справочника.

**Параметры**

- **block_id**: Optional[str]
  - Идентификатор блока справочника, из которого получены данные
- **dict_id**: Optional[str]
  - Идентификатор справочника, из которого получены данные
- **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника, из которого получены данные
- **binding**: Dict
  - Словарь с привязками данных справочника
- **attributes**: CAttributes
  - Метаданные всех атрибутов справочника

**Свойства**

- count -> int
  - Получение общего количества элементов в справочнике


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

### Методы

Сериализация

```
serialize(self) -> Dict
```

Сериализация атрибутов и данных справочника в словарь вида:

```
{
   "binding": {},
   "fields": [],
   "data": []
}
```

Очистка класса

```
clear(self) -> None
```

Очистка класса справочника, для его корректного сборщиком мусора.

Получение элемента по его индексу

```
__getitem__(idx: int) -> CDictionaryElement
```

```
elements = CDictionaryElements(...)
element = elements[idx]
```

Получение элемента по его индексу.

В случае иерархического справочника - создается плоский список и выбирается элемент.

- **idx** - индекс элемента, начиная с нуля.

Заполнение элементов справочника

```
fill(raw_data: Dict) -> None
```

Заполнение данных справочника

```
fill_data(raw_data: List[Union[List, Dict]], clear: bool = False) -> None:
```

Получение элементов справочника

```
get_elements(
   flatten: bool = False
) -> List[Union[CDictionaryElement, CDictionaryHierarchyElement]]:
```

Получение массива с данными справочника

```
to_json(self) -> List[Dict]
```

Получение объектов справочника (устаревший метод)

```
to_old_format_json(flatten: bool = False) -> List[Dict]
```

Получение элемента по первичным ключам

```
get_element_by_primary_keys(
   keys: Dict
) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

Получение оригинального элемента по элементу альтернативной иерархии

```
get_element_from_alt_hierarchy_element(
    alt_hierarchy_element: Union[CDictionaryElement, CDictionaryHierarchyElement]
) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]
```

Получение оригинального элемента по объекту альтернативной иерархии (устаревший метод)

```
get_element_from_alt_hierarchy_element_old_format(
    alt_hierarchy_element: Dict
) -> Union[None, CDictionaryElement, CDictionaryHierarchyElement]
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionary

Основной класс для работы со справочниками. Создает окружение и отправляет запросы на шину/API для создания, обновления, удаления и расчета справочника, а так же данных справочника.

После завершения работы со справочником необходимо вызвать метод .clear()

**Импорт**

```
from src.service_dictionary import CDictionary
```

**Инициализация**

```
CDictionary(
   block_id: Optional[str] = None,
   user_id: Optional[str] = None,
   dictionary_id: Optional[str] = None,
   dictionary_type: Optional[Union[EDictionaryTypes, str]] = None,
   attributes: Optional[CAttributes] = None,
   revision_id: Optional[str] = None,
   lazy_load: bool = False,
   binding: Optional[Dict] = None,
   specific_params: Optional[Dict] = None
)
```

- **block_id** - идентификатор блока справочника.
- **user_id** - идентификатор пользователя.
- **dictionary_id** - идентификатор справочника. Если не указывать, то ищется автоматически, исходя из **block_id**.

  Если по блоку будет найдено несколько справочников, то будет использован идентификатор первого найденного.

- **dictionary_type** - тип справочника. Если не указывать, то ищется автоматически, исходя из **block_id**.
- **revision_id** - идентификатор ревизии данных справочника. Если не указывать, то для работы с данными будет использована самая свежая ревизия.

Следующие параметры позволяют взаимодействовать не только с данными справочника, но и с его структурой.

- **attributes** - атрибуты справочника. Если не указывать, то ищутся автоматически, исходя из **dictionary_id**.


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- **lazy_load** - флаг отложенной загрузки данных справочника. Если включен, то справочник не будет обновлять данные при расчете блока.
- **binding** - привязки атрибутов справочника.
- **specific_params** - специфические параметры, необходимые для расчета справочника.

```
execution_context.create_dictionary_instance(
   dictionary_id: str,
   revision: Optional[str] = None,
   blockid: Optional[str] = None,
)
```

### Параметры

- **block_id**: Optional[str]
  - Идентификатор блока, в котором создан справочник
- **user_id**: Optional[str]
  - Идентификатор пользователя, для которого создан справочник
- **dictionary_type**: EDictionaryTypes
  - Тип справочника
- **attributes**: CAttributes
  - Список атрибутов справочника
- **binding**: Dict
  - Словарь с привязками полей справочника
- **lazy_load**: bool
  - Флаг ленивой (отложенной) загрузки данных справочника
- **dictionary_id**: Optional[str]
  - Идентификатор справочника
- **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника
- **specific_params**: Dict
  - Специфические параметры для инициализации конкретного типа справочника
- **hierarchy**: Optional[CDictionaryHierarchyManager]
  - Иерархии справочника

### Методы

Сериализация

```
serialize(self) -> Dict
```

Сериализация описания справочника в словарь вида:

```
{
   "dictionary_id",
   "dictionary_type",
   "revision_id",
   "block_id",
   "user_id",
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

```
    "binding",
    "lazy_load",
    "specific_params"
}
```

Очистка класса

```
clear(self) -> None
```

Очистка класса справочника, для его корректного сборщиком мусора.

Инвалидация кеша

```
invalidate_cache(self, by_block: bool = False) -> None
```

Инвалидация кеша справочника.

*Работа с данными*

Получение данных справочника

```
get_data(
    limit: Optional[int] = None,
    offset: Optional[int] = None,
    get_by_block: bool = False
) -> CDictionaryElements
```

Получение рассчитанных данных справочника. Если в справочнике есть иерархия - придут иерархические данные.

- **limit** - количество объектов.
- **offset** - сдвиг объектов.
- **get_by_block** - получение данных по идентификатору блока.

Получение исходных данных справочника

```
get_origin_data(
    limit: Optional[int] = None,
    offset: Optional[int] = None,
    get_by_block: bool = False
) -> CDictionaryElements
```

Получение данных справочника без иерархий.

- **limit** - количество объектов.
- **offset** - сдвиг объектов.
- **get_by_block** - получение данных по идентификатору блока.

Получение отфильтрованных данных справочника

```
get_filter_data(
    filter_params: List[Dict],
    limit: Optional[int] = None,
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

```
    offset: Optional[int] = None,
    get_by_block: bool = False
) -> CDictionaryElements
```

Получение данных справочника с фильтрацией элементов. Можно фильтровать как по ключам, так и по значениям. Если фильтр будет пустой, то в ответе вернется пустой массив.

- **filter_params** - параметры фильтра. Формат:

  ```
  [{"key": <key>, "eq": <eq>, "vl": <vl>}]
  ```

  где:

  - <key> - ключ атрибута, по которому производится фильтрация;
  - <eq> - условие фильтрации. На текущий момент реализованы следующие условия:
    - **==** - равно
    - **!=** - не равно
    - **in** - входит в массивю
    - **not in** - не входит в массив.
  - <vl> - значения ключа для фильтра.
    - Для корректной передачи массивов необходимо использовать `in/not in`
- **limit** - количество объектов.
- **offset** - сдвиг объектов.
- **get_by_block** - получение данных по идентификатору блока.

*Работа с ревизиями (дельтами)*

Получение последней ревизии

```
get_last_delta() -> Optional[str]
```

Получение последнией созданной ревизии для справочника. Если ревизии отсутсвуют - придет None.

Получение списка всех ревизий

```
get_all_deltas() -> List[Dict]
```

Получение списка всех ревизий для справочника. Формат ответа:

```
[
    {
      "revision": "<идентификатор ревизии>",
      "prev_revision": "<идентификатор предыдущей ревизии>",
      "delta": {
        "insert": "<Количество добавленных элементов>",
        "update": "<Количество измененных элементов>",
        "delete": "<Количество удаленных элементов>"
      }
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

```
    }
]
```

Получение истории ревизии

```
get_revision_history() -> List[Dict]
```

Получение списка ревизий от начальной (созданной при создании справочника) до текущей (указывается в параметрах класса). Формат ответа:

```
[
    {
      "revision": "<идентификатор ревизии>",
      "prev_revision": "<идентификатор предыдущей ревизии>",
      "delta": {
        "insert": "<Количество добавленных элементов>",
        "update": "<Количество измененных элементов>",
        "delete": "<Количество удаленных элементов>"
      }
    }
]
```

Сохранение дельты

```
save_delta(delta: Dict) -> Optional[str]
```

Сохранение дельты справочника и создания новой ревизии от текущей (указывается в параметрах класса).

Данные в самом справочнике не будут затронуты изменениями.

В ответе придет идентификатор созданной ревизии.

Ревизии уникальны для блока, создающего их, и идентификатора предыдущей ревизии, то есть, в случае, если один блок будет пытаться сохранить несколько ревизий от одной и той же, то результатом будет одна ревизия, сохраненная последней (остальные данные будут затерты)

- **delta** - Словарь с изменениями данных, формата

  {<insert>, <update>, <delete>}

  - <insert>, <update>, <delete> являются массивами, содержащими словари формата:
    - {"keys": {<id>: <value>}, "value": {<id>: <value>}}
      - **keys** - словарь ключей (атрибутов, у которых параметр primary_key=1);
      - **value** - словарь всех атрибутов (в том числе и ключей);
      - **<id>** - идентификатор атрибута;
      - **<value>** - значение атрибута.


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

*Работа с потоковым справочником*

Следующие методы работают только с потоковым справочником. Все остальные типы справочников вызовут ошибку.

Загрузка данных

load_stream_data(insert: List, update: List, delete: List) -> Dict

Загрузка (добавление, обновление или удаление) элементов в потоковый справочник.

После загрузки элементов, все ревизии справочника будут удалены (т.к. данные справочника были изменены).

- **insert**: Массив словарей с добавленными элементами, формат:
  - {"keys": {<id>: <value>}, "value": {<id>: <value>}}
    - **keys** - словарь ключей (атрибутов, у которых параметр primary_key=1);
    - **value** - словарь всех атрибутов (в том числе и ключей);
    - **<id>** - идентификатор атрибута;
    - **<value>** - значение атрибута.
- **update**: Массив словарей с обновленными элементами, формат аналогичен insert.
- **delete**: Массив словарей с удаленными элементами, формат аналогичен insert.
  - Словарь value можно не указывать, т.к. удаление происходит по ключам объекта.

Ответ:

```
{
   "insert": <Массив со значениями 0/1, 0 - если сохранение успешно, 1 - если нет.>
   "update": <Массив со значениями 0/1, 0 - если сохранение успешно, 1 - если нет.>,
   "delete": <Массив со значениями 0/1, 0 - если сохранение успешно, 1 - если нет.>
}
```

Очистка потокового справочника

clear_stream_data() -> bool

Очистка потокового справочника от данных.

Все ревизии, если они были, будут так же удалены.

В ответе придет флаг успешности очистки.


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryAlternativeHierarchy

Класс по работе с альтернативной (иерархия из других справочников, связанных с оригинальным по ключу какого-либо поля) иерархией справочника.

Наследуется от CDictionaryHierarchy

**Импорт**

```
from src.service_dictionary.hierarchy.c_dictionary_alternative_hierarchy import CDictionaryAlternativeHierarchy
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он приходит при получении иерархии справочника

```
CDictionaryAlternativeHierarchy(
    dictionary_id: str,
    user_id: Optional[str] = None,
    block_id: Optional[str] = None,
    revision_id: Optional[str] = None,
    name: str = "",
    key: str = "",
    block: str = "",
    block_key: str = "",
    revision: Optional[str] = None
)
```

- **dictionary_id** - Идентификатор справочника, для которого строится иерархия
- **user_id** - Идентификатор пользователя справочника, для которого строится иерархия
- **block_id** - Идентификатор блока справочника, для которого строится иерархия
- **revision_id** - Идентификатор ревизии данных справочника, для которого строится иерархия
- **name** - Наименование иерархии
- **key** - Ключ атрибута, отвечающего за иерархию
- **block** - Идентификатор блока с альтернативной иерархией
- **block_key** - Ключ атрибута в блоке с альтернативной иерархией (для связи ключей)
- **revision** - Ревизия данных блока с альтернативной иерархией

**Параметры**

Наследуются от CDictionaryHierarchy, кроме этого добавляется:

- **key**: str
    - Ключ атрибута, отвечающего за иерархию
- **alt_block**: str
    - Идентификатор блока с альтернативной иерархией
- **alt_block_key**: str
    - Ключ атрибута в блоке с альтернативной иерархией для связи ключей
- **alt_revision**: str


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

- Ревизия данных блока, для которой отображаем иерархию

**Методы**

Наследуются от CDictionaryHierarchy


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryBasicHierarchy

Класс по работе с базовой (родитель-ребенок) иерархией справочника.

Наследуется от CDictionaryHierarchy

**Импорт**

```
from src.service_dictionary.hierarchy.c_dictionary_basic_hierarchy import CDictionaryBasicHierarchy
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он приходит при получении иерархии справочника

```
CDictionaryBasicHierarchy(
   dictionary_id: str,
   user_id: Optional[str] = None,
   block_id: Optional[str] = None,
   revision_id: Optional[str] = None
)
```

- **dictionary_id** - Идентификатор справочника, для которого строится иерархия
- **user_id** - Идентификатор пользователя справочника, для которого строится иерархия
- **block_id** - Идентификатор блока справочника, для которого строится иерархия
- **revision_id** - Идентификатор ревизии данных справочника, для которого строится иерархия

**Параметры**

Наследуются от CDictionaryHierarchy

**Методы**

Наследуются от CDictionaryHierarchy


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

Класс CDictionaryElement

Класс для взаимодействия с массивом объектов справочника.

Наследуется от CAttributes, но тут учитываются и значения атрибутов справочника.

После завершения работы необходимо вызвать метод .clear()

**Импорт**

```
from src.service_dictionary.elements import CDictionaryElement
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он приходит при получении атрибутов справочника

```
CDictionaryElement(
   elements: Optional[List] = None,
   fields: Optional[List[Dict]] = None
)
```

- **elements** - массив со значениями атрибутов.
- **fields** - массив с описанием атрибутов.

Можно не передавать атрибуты при инициализации, а добавить их позднее, методами **add_attributes** или **refresh_attributes**.

**Параметры**

Наследуются от CAttributes.

**Свойства**

Наследуются от CAttributes. К ним так же добавляются:

- **hash_primary_key**-> int
    - Получение хеша первичного ключа. Хеш берется по словарю

      {<id>: <value>}

**Методы**

Наследуются от CAttributes. К ним так же добавляются:

Сериализация

```
serialize() -> Dict
```

Сериализация атрибутов и данных справочника в словарь вида:


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

```
{
   "elements": {},
   "fields": []
}
```

Где **elements** сериализуются методом to_json()

а **fields** - CAttributes.serialize()

Сравнение первичных ключей по хешу

```
compare_primary_keys(keys: Dict) -> bool
```

Заполнение класса с использованием созданного CAttributes

```
fill_from_attributes(attributes: CAttributes, elements: Optional[List] = None) -> None
```

Получение словаря с данными справочника

```
to_json() -> Dict
```

Получение объекта справочника (устаревший метод)

```
to_old_format_json(
   binding: Optional[Dict] = None,
   block_id: Optional[str] = None,
   dict_id: Optional[str] = None,
   revision_id: Optional[str] = None,
) -> Dict:
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryHierarchy

Класс по работе с иерархией справочника.

Данный класс является абстрактрым и используются, наследованные от него классы CDictionaryBasicHierarchy и CDictionaryAlternativeHierarchy

После завершения работы со справочником необходимо вызвать метод .clear()

**Параметры**

- **name**: str
  - Наименование иерархии
- **basic**: bool
  - Флаг того, что иерархия является базовой (родитель - ребенок)
- **default**: bool
  - Флаг иерархии по умолчанию
- **dictionary_id**: str
  - Идентификатор справочника, для которого строится иерархия
- **user_id**: Optional[str]
  - Идентификатор пользователя справочника, для которого строится иерархия
- **block_id**: Optional[str]
  - Идентификатор блока справочника, для которого строится иерархия
- **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника, для которого строится иерархия

**Методы**

Очистка класса

```
clear(self) -> None
```

Очистка класса справочника, для его корректного сборщиком мусора.

Получение данных справочника

```
get_data(limit: Optional[int] = None, offset: Optional[int] = None, get_by_block: bool = False) -> CDictionaryElements
```

Получение дочерних элементов справочника

```
get_child_elements(parent: Union[CDictionaryHierarchyElement, str], all_child: bool = False) -> CDictionaryElements
```

Получение родительских элементов справочника

```
get_parent_element(parent: Union[CDictionaryHierarchyElement, str]) -> CDictionaryElements
```

Получение элементов по уровню

```
get_elements_on_level(level: int) -> CDictionaryElements
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryHierarchyElement

Класс для взаимодействия с массивом объектов справочника.

Наследуется от CDictionaryElement, но тут учитываются иерархия элементов

После завершения работы необходимо вызвать метод .clear()

**Импорт**

```
from src.service_dictionary.elements import CDictionaryHierarchyElement
```

**Инициализация**

Аналогично CDictionaryElement.

**Параметры**

Наследуются от CAttributes. К ним так же добавляются:

- **child**: List[CDictionaryHierarchyElement]
  - Массив дочерних элементов объекта
- **level**: int
  - Уровень объекта в иерархии. Начинается с нуля

**Свойства**

Наследуются от CDictionaryElement. К ним так же добавляются:

- **parent_key** -> CAttribute
  - Получение атрибута, отвечающего за родителя в базовой иерархии.
- **hierarchy_key** -> CAttribute
  - Получение атрибута, отвечающего за уникальный ключ для связи базовой иерархии.

**Методы**

Наследуются от CDictionaryElement. К ним так же добавляются:

Получение массива с данными справочника

```
to_json() -> List[Dict]
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

# Класс CDictionaryHierarchyManager

Класс по работе с иерархиями справочника.

Инициализируется при вызове CDictionary.

**Импорт**

```
from src.service_dictionary.hierarchy import CDictionaryHierarchyManager
```

**Инициализация**

Отдельно инициализировать класс не нужно, т.к. он инициализируется при вызове справочника

```
CDictionaryHierarchyManager(
   dictionary_id: str,
   user_id: Optional[str] = None,
   block_id: Optional[str] = None,
   revision_id: Optional[str] = None
)
```

- **dictionary_id** - Идентификатор справочника, для которого строится иерархия
- **user_id** - Идентификатор пользователя справочника, для которого строится иерархия
- **block_id** - Идентификатор блока справочника, для которого строится иерархия
- **revision_id** - Идентификатор ревизии данных справочника, для которого строится иерархия

**Параметры**

- **dictionary_id**: str
  - Идентификатор справочника, к которому подключен менеджер
- **user_id**: Optional[str]
  - Идентификатор пользователя справочника, к которому подключен менеджер
- **block_id**: Optional[str]
  - Идентификатор блока справочника, к которому подключен менеджер
- **revision_id**: Optional[str]
  - Идентификатор ревизии данных справочника, к которому подключен менеджер

**Свойства**

- **count** -> [int]
  - Количество иерархий
- **items** -> List[Dict]
  - Список всех иерархий в формате

    {<name>, <type>, <default>}

- **default** -> Union[CDictionaryBasicHierarchy, CDictionaryAlternativeHierarchy]


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)

    - Получение иерархии по умолчанию (либо первой в списке, если по умолчанию не было отмечено иерархий)
- **basic** -> Optional[CDictionaryBasicHierarchy]
    - Получение базовой иерархии (родитель - ребенок), если она есть
- **enabled** -> bool
    - Флаг наличия иерархий у справочника

### Методы

Очистка класса

```
clear(self) -> None
```

Очистка класса справочника, для его корректного сборщиком мусора.

Заполнение иерархий

```
fill_from_attributes(attributes: CAttributes) -> None
```


© ООО «Бюро Информационных Технологий» (ООО «БИТ»)