**119270, Москва, Лужнецкая наб., д. 6,**
**стр.1, офис 214, ООО «ЭР СИ О»**
**Тел./факс: (495) 287-98-87**
**E-mail: info@rco.ru**
**http://www.rco.ru**

# Руководство разработчика

# RCO Address Parser

Версия 3.0

Москва, 2021

В содержание данного документа могут быть внесены изменения без предварительного уведомления. Названия организаций, имена и даты, используемые в качестве примеров, являются вымышленными, если не оговорено обратное.

# Содержание

# Введение

Программный продукт RCO Address Parser 3.0 (далее RCO AP3) предназначен для разбора и нормализации российских и иностранных почтовых адресов. Для адресов РФ реализованы сопоставление с классификатором ФИАС, представление строки адреса в унифицированном виде, выдача списка подсказок-вариантов для продолжения адреса. Поддерживаются также адреса мест рождения, включая адреса СССР. Входными данными является текстовая строка с адресом или набор полей с адресными элементами, передаваемые по протоколу SOAP. Входные данные разбираются, и структурированный результат разбора адреса возвращается в виде JSON, внутри SOAP ответа.

Архитектура сервиса обеспечивает работу сервиса без использования СУБД. Если пользовательское приложение работает с СУБД Oracle, то оно может использовать пакеты, входящий в состав поставки, облегчающие обмен данными с RCO AP3.

RCO AP3 позволяет обрабатывать как структурированные, так и неструктурированные адреса. Существует возможность настройки коррекции строки адреса перед началом обработки. Словари для распознавания адресов включают в себя места рождения СССР, правила разбора строки помещения.

Настоящее руководство предназначено для разработчиков и представляет собой руководство по настройке взаимодействия между клиентским программным комплексом и RCO AP3. Документ включает описание структуры RCO AP3 и входящий в него модулей; описание протокола обмена, включая поля запроса и ответа; описание их ресурсов и API взаимодействия, описание работы библиотек в составе RCO AP3 и их функций; описание функций пакетов, при работе с RCO AP3 из СУБД Oracle и другие технические данные.

Предполагается, что читатель обладает навыками программирования и владеет знаниями сетевых технологий (HTTP, SOAP протоколы) и форматов (XML, JSON). Если предполагается использование продукта совместно с СУБД Oracle, то для этого потребуются знания и навыки программирования под Oracle.

# Описание программы

Программный продукт RCO AP3 состоит из следующих компонентов:

- Службы разбора адресов, далее Службы, настроенной на использование модуля разбора адресов;
- Тестового Web−приложения, развёртываемого под Internet Information Services (IIS), позволяющего осуществить ручной ввод адреса или адресных элементов,

отправить его по SOAP протоколу в сервис разбора и представить полученные по SOAP результаты разбора, включая подсказки ввода адреса, в окне браузера.

- Пакета Oracle для работы с сервисом разбора из автоматизированных средств, использующих Oracle Database (также по SOAP протоколу);
- Скриптов, позволяющих проводить обновление базы ФИАС с сайта https://fias.nalog.ru/Updates.

В свою очередь, служба разбора адресов в составе RCO AP3 включает следующие модули:

- Адаптер службы разбора адресов, далее Адаптер AP3, отвечающий за сетевое взаимодействие по протоколу SOAP и вызов модуля разбора адресов при обработке запросов с адресной информацией (устанавливается в качестве сервиса Windows);
- Модуль разбора адресов;
- Утилиту для тестирования модуля разбора адресов через прямое обращение FXDirectClient_x64.exe, далее Прямой клиент, позволяющую инициализировать модуль разбора, использовать непосредственно его методы, произвести разбор адреса и сохранить в виде файла результаты (Прямой клиент и Адаптер AP3 осуществляют обмен данными с модулем разбора в формате JSON);
- Утилиту для тестирования работы Службы разбора адресов через обращение по сети FXClient.exe, далее Сетевой клиент (использует SOAP протокол);
- набор тестовых файлов для утилит тестирования.

Модуль разбора представляет из себя dll библиотеку, предоставляющую методы для обработки адресов в виде JSON запросов и загружаемую по требованию Адаптера AP3 или Прямого клиента.

Схема взаимодействия компонентов RCO AP3 приведена на рис. 1.

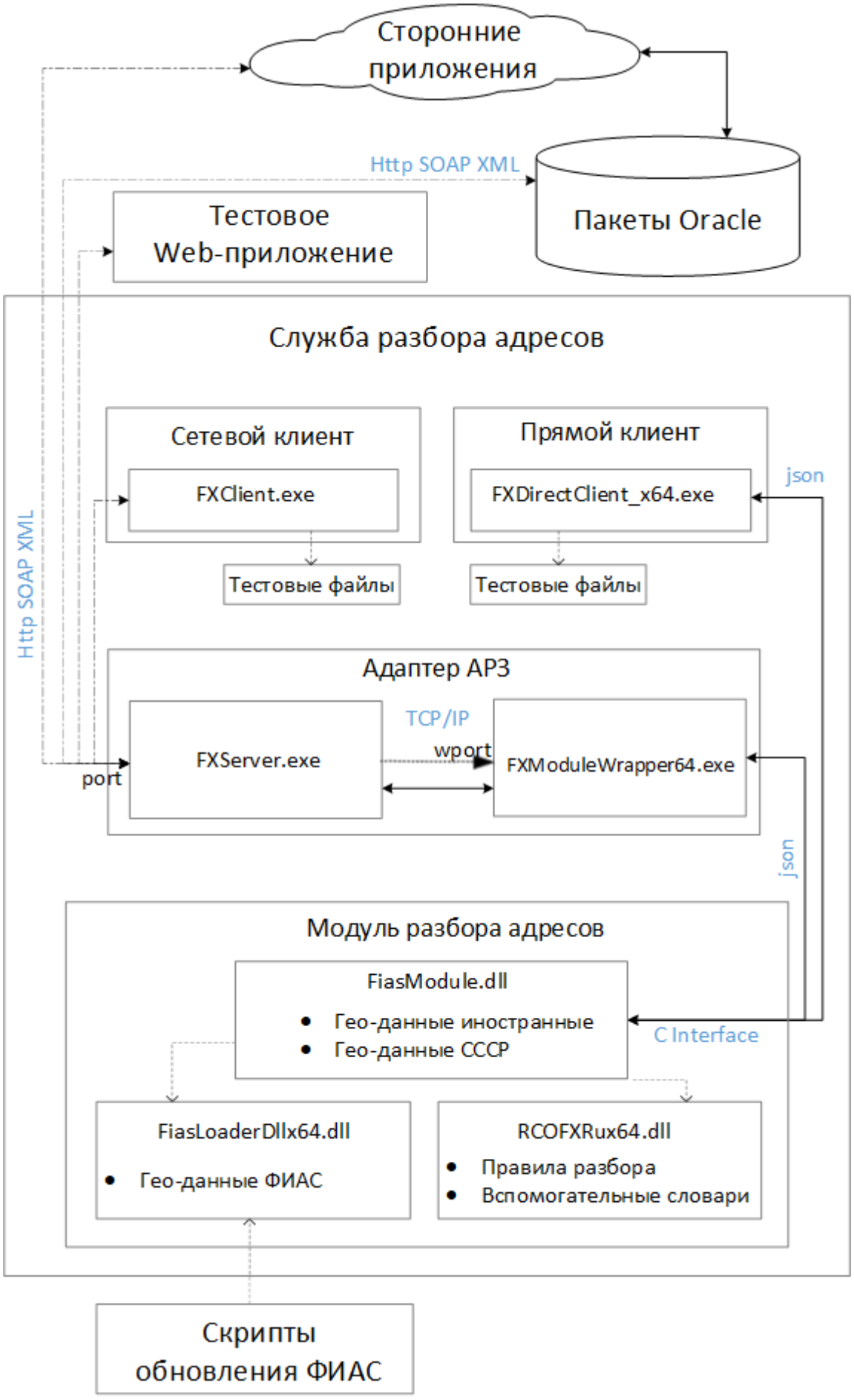


Рис. 1. Схема взаимодействия компонентов RCO АРЗ

# Программный интерфейс Службы разбора адресов

## Общее описание

Служба разбора адресов обслуживает запросы по протоколу SOAP. Запрос на обработку адреса принимается в качестве содержимого конверта SOAP в формате JSON (кодировка UTF-8). Аналогичным образом возвращается результат обработки.

За реализацию протокола SOAP отвечает Адаптер АРЗ. Компонент Адаптера АРЗ FXServer.exe принимает запросы через SOAP XML. Запрос в формате JSON должен быть закодирован в BASE64. Адаптер взаимодействует с модулями с помощью промежуточного процесса FXModuleWrapper, который непосредственно вызывает библиотеку Модуля разбора адресов, передавая ей JSON запрос с одним или несколькими адресами, представленными в виде строк или наборов полей.

Модуль разбора адресов поставляется в виде 64bit библиотеки FiasModule.dll, использующей FiasLoaderDllx64.dll, специализированную библиотеку для работы с базой ФИАС, а также библиотеку RCOFxRux64.dll для выделения адресных элементов из строки адресе, и предназначен для развёртывания в качестве плагина для Адаптера АРЗ.

## Перечень используемых ресурсов

### Лингвистические ресурсы

Вместе с адресными ресурсами основной библиотеки FiasLoader модуль загружает также лингвистическую библиотеку «RCOFXRux64.dll», предназначенную для первичного разбора адресов заданных единой текстовой строкой. Лингвистическая библиотека использует правила разбора адресной строки и словари элементов, содержащиеся в каталоге «./dic/», файлы из каталога грузятся в соответствии с файлом конфигурации «./config.xml», логирование и вывод промежуточных результатов разбора адресных строк регламентируется файлом «./fx.ini».

### Справочник иностранных адресных элементов

Для определения принадлежности адреса иностранным государствам модуль загружает краткий справочник стран и городов мира, с кодами стран из файла «./foreign/addr-foreign-obj.tsv». Проверка осуществляется упрощённой токенизацией адресной строки и оценкой количества не омонимичных российским вхождений полученных токенов в словарь иностранных наименований. Для помеченных как иностранные адресов выставляется код ОКСМ, а поиск по строке не производится.

### Иерархический справочник адресов бывшего СССР

Для определения принадлежности адреса (рождения) бывшим республикам СССР модуль содержит иерархические словари «./birth/USSR_city.txt» и «./birth/USSR_subject.txt».

## Описание процесса обработки запроса

В первую очередь, для каждого адресного запроса JSON производится проверка на корректность. Корректный запрос должен содержать хотя бы один из следующих элементов:

- GUID ФИАС адресного элемента (города/улицы) или дома;
- Код ФИАС адресного элемента;
- Не пустая адресная строка;
- Набор типизированных полей адреса.

Если таких элементов в запросе нет, выдаётся ошибка «"No GUID, Code, AddressLine of FieldSet found in request"».

Затем, при наличии ненулевого GUID, производится поиск среди адресных элементов или домов по GUID с восстановлением полного адреса вверх по иерархии ФИАС. Найденный адрес выдаётся в качестве результата. Если GUID указан, но в ФИАС он не значится, выдаётся ошибка «Code search failed for FiasGUID "GUID"».

Если вместо GUID указан иерархический код ФИАС, то производится аналогичный поиск по коду.

Если задана адресная строка, то производится проверка на наличие в ней названий иностранных адресных элементов. Если из сработавших государств какое-либо набрало число элементов выше двух или число элементов страны больше числа Российских синонимов в строке, адрес признаётся иностранным. Проставляется код OKSM в результатах, сбрасывает флаг СССР, если страна в него не входила, проводится базовая нормализация строки.

Если не заданы отдельные поля адреса, но задана единая строка, строка разбирается при помощи FX на адресные элементы. Если в итоге никаких частей адреса не выделено, возвращается ошибка «"No valid Address request fields given or parsed from address line».

Если в исходном запросе стоял флаг места рождения, и проверка на иностранный адрес не выявила другую страну, проводится поиск полей адреса по географическим объектам СССР. Для найденных адресов проводится предварительная нормализация по

базе СССР, присваивается OKSM из найденной республики. Если адрес принадлежал РСФСР и ему поставлен в соответствие код ФИАС, этот код используется для нормализации.

Наконец, если адрес всё ещё считается российским, производится основной поиск по базе ФИАС.

# Описание входных данных

## Общий формат JSON запроса

В случае запроса на разбор единичного адреса:

```
{
	"address_id": " id1",
	"name": "value",
	…
}
```

В случае мульти-запроса на разбор серии адресов:

```
{
	"id1":	{
		"query_field_name": "value",
		…
	},
	"id2":	{
		"query_field_name": "value",
	…
	},
}
```

## Описание полей JSON запроса

| Поле | Тип JSON | Описание |
|---|---|---|
| address_id | Строка | Строковый идентификатор запроса |
| is_birth_address | Булевое | Проводить ли поиск адреса в адресах СССР |
| try_fuzzy | Булевое | Использовать ли нечёткое сравнение названий адресных элементов |
| address_line | Булевое | Адресная строка с индексом, городом, улицей, домом, помещением |
| fias_code | Число | Готовый ФИАС код адресного объекта (АО) 4200001500000000000 |
| guid | Строка | GUID АО или дома "bd02ecac-620d-4517-aae3-a56b6da5ca41" |
| fields_dic{} | Словарь | Словарь полей, заменяющих «address_line» |
| Address:Part_Country | Строка | Страна, если не указано, предполагается Россия |
| Address:Part_Subject | Строка | Субъект РФ |
| Address:Part_Region | Строка | Район |
| Address:Part_City | Строка | Город/Поселение |
| Address:Part_Street | Строка | Улица |
| Address:house_line | Строка | Строка с домом, корпусом и строением |
| Address:flat_line | строка | Строка с подъездом, этажом, квартирой/помещением и комнатой |

## Пример запроса с адресной строкой

```
{
	"address_id":"123456",
	"address_line":"г Москва, Лужнецкая набережная, д 6 стр 1, кв 214",
	"fields": []
}
```

## Пример запроса с полями

```
{
	"address_id":"12345",
	"fields_dic": {
```

```
                "Address:Part_Country": "РОССИЯ",
                "Address:Part_Subject": "Московская область",
                "Address:Part_Region": "ДОМОДЕДОВСКИЙ р-н",
                "Address:Part_City": "ДОМДЕДОВО г",
                "Address:Part_Street": "АВИАЦИОННАЯ  ул",
                "house_line": "Д 1 КОРП А",
                "flat_line": "КВАРТИРА 6"
        },
        "is_birth_address": false,
        "try_fuzzy": true
}
```


# Описание выходных данных

## Общий формат JSON ответа

Ответ даже для одиночного запроса всегда выдаётся как на мульти-запрос, со словарём по id:

```
{
        "id1": {
                "address_id": "id1",
                "res_field_name": "value",
                …
        }
}
```

## Описание полей JSON ответа

| Название | Тип | Описание |
|---|---|---|
| address_id | Строка | Идентификатор адресной строки запроса (полезен при обработке мульти-запросов). |
| address_line | Строка | Повторение строки из запроса |
| GUID | Строка | Повторение GUID из запроса |
| fias_code | Число | Повторение кода ФИАС из запроса |
| kladr_code | Число | Повторение кода КЛАДР из запроса |
| error | Строка | Строка с ошибкой, если разбор запроса или поиск не удались. |
| russian | Булевое | Флаг ставится если адрес российский |
| USSR | Булевое | Флаг ставится если адрес из бывшего СССР |
| OKSM | Число | Целочисленный код страны по ОКСМ |
| fields[{},{}] | Словарь | Повтор полей из запроса или результат разбора адресной строки на элементы |
| name | Строка | Название элемента адреса |
| key | Строка | Ключ элемента адреса |
| Type | Строка | Тип элемента адреса в соответствии с правилами Cape |
| fields_norm[{},{}] | Словарь | Поля с нормализованным именем/ключом |
| Name | Строка | Название элемента адреса |
| Key | Строка | Ключ элемента адреса |
| Type | Строка | Тип элемента адреса в соответствии с правилами Cape |
| norm_parts_line | Строка | Составленная из нормализованных элементов строка |
| Variants[{…},{…}] | Список | Список различных вариантов разбора адреса |
| norm_line | Строка | Нормализованная адресная строка варианта |
| norm_line_lev_dist | num | Процент совпадения строки варианта «norm_line» с «norm_parts_line» |
| PART_MATCHES | str | Строка формата «([0-9_]{1}[*h]?,)+», даёт позиции входных частей адреса, найденных в данном варианте разбора; * - часть адреса сопоставлена нечётким поиском, h – по истории переименований. |
| FIAS_Parts[{},{}] | list | Список адресных элементов ФИАС для текущего варианта разбора |
| FIAS_Name | str | Поле «FORMALNAME» адресного элемента в ФИАС |
| FIAS_Socr | str | Поле «SHORTNAME» адресного элемента в ФИАС |
| FIAS_Level | num | Уровень элемента в иерархии ФИАС |
| FIAS_Code | num | Иерархический код ФИАС |
| FIAS_GUID | str | GUID адресного объекта варианта |
| FIAS_Code | num | Иерархический код ФИАС адресного объекта варианта |
| FIAS_Level | num | Уровень адресного объекта варианта |
| KLADR_Code | num | Иерархический код КЛАДР адресного объекта варианта |

| | | |
|---|---|---|
| OKATO | num | Код OKATO адресного объекта варианта |
| OKTMO | num | Код OKTMO адресного объекта варианта |
| address_post_index | num | Почтовый код адресного объекта варианта |
| house_match | str | exact – единственный вариант дома/multiple – множество вариантов/none – дом не задан |
| flat_match | str | exact – единственный вариант квартиры/multiple – множество вартов/none – не задано |
| house_variants | dct | |
| house_line | str | Нормализованная строка с домом, корпусом и строением |
| post_index | num | Почтовый код адресного дома |
| HOUSE_GUID | str | ФИАС GUID дома |
| appartment | str | Нормализованная строка с квартирой/помещением |

## Пример обращения к Службе

Запрос:

```
{
        "address_id":"123456",
        "address_line":"балашиа, некраова 4 59",
        "fields":[],
        "is_birth_address":false,
        "try_fuzzy":true
}
```

Ответ:


```
{
  "123456": {
    "address_id": "123456",
    "guid": "00000000-0000-0000-0000-000000000000",
    "fias_code": 0,
    "kladr_code": 0,
    "address_line": "балашиа, некраова 4 59",
    "russian": true,
    "USSR": true,
    "OKSM": 643,
    "fields": [
      {
        "name": "балашиа",
        "type": "Address:Unknown"
      },
      {
        "name": "некраова",
        "type": "Address:Unknown"
      },
      {
        "name": "4",
        "type": "Address:Part_House"
      },
      {
        "name": "59",
        "type": "Address:Part_Apartment"
      }
    ],
    "fields_norm": [
      {
        "name": "4",
        "type": "Address:Part_House"
      },
      {
        "name": "59",
        "type": "Address:Part_Apartment"
      },
      {
        "name": "БАЛАШИА",
        "type": "Address:Unknown"
      },
      {
        "name": "НЕКРАОВА",
        "type": "Address:Unknown"
      }
    ],
    "norm_parts_line": "БАЛАШИА, НЕКРАОВА, ДОМ 4, КВ 59",
    "variants": [
      {
        "norm_line": "ОБЛ МОСКОВСКАЯ, Г БАЛАШИХА, УЛ НЕКРАСОВА, ДОМ 4, КВ 59",
        "norm_line_lev_dist": 42,
        "PART_MATCHES": "_,2*,3*",
        "FIAS_Parts": [
```

```
            {
              "FIAS_Name": "МОСКОВСКАЯ",
              "FIAS_Socr": "ОБЛ",
              "FIAS_Level": 1,
              "FIAS_Code": 5000000000000000000
            },
            {
              "FIAS_Name": "БАЛАШИХА",
              "FIAS_Socr": "Г",
              "FIAS_Level": 4,
              "FIAS_Code": 5000003600000000000
            },
            {
              "FIAS_Name": "НЕКРАСОВА",
              "FIAS_Socr": "УЛ",
              "FIAS_Level": 7,
              "FIAS_Code": 5000003600000000004
            }
          ],
          "FIAS_GUID": "925e4f85-36af-497b-8868-8172c30a37e5",
          "FIAS_Code": 5000003600000000004,
          "FIAS_Level": 7,
          "KLADR_Code": 500000360000004,
          "KLADR_Level": 6,
          "OKATO": 46404000000,
          "OKTMO": 46704000001,
          "house_variants": [
            {
              "house_line": "ДОМ 4",
              "post_index": 143900,
              "HOUSE_GUID": "f1fff92f-54b9-4ec2-8567-d9e68a5da6d2",
              "appartment": "КВ 59",
              "appartment_matched": true
            }
          ],
          "address_post_index": 143900,
          "house_match": "exact",
          "flat_match": "exact"
        }
      ]
    }
}
```

# Программный интерфейс библиотеки загрузки и поиска по данным ФИАС

## Общая схема работы с библиотекой.

Библиотека FiasLoaderDllx64.dll предназначена для эффективной работы с данными ФИАС через низкоуровневый программный интерфейс. Ее функциональность опосредованно доступна через обращения к Службе разбора адресов. В подавляющем большинстве случаев обращаться к данной библиотеке нет необходимости.

Для начала работы необходимо загрузить в библиотеку данные ФИАС. Загрузка производится вызовом функции **`loadFiasAO`**, см. Описание функций библиотеки.

В качестве параметров библиотеке передаётся путь к конфигурационному файлу. Загрузка при этом произойдёт быстрей, используя кэш, если данные этой конфигурации уже загружались ранее.

Библиотека является многопоточной и для каждого потока, вызывающего инициализацию, выдаст свой дескриптор (handle).

После загрузки по установленному дескриптору можно посылать адресные запросы. Для этого библиотека предоставляет функции, начинающиеся с **`search`**….

Результаты поиска можно получить при помощи **`get`**… функций. Для каждого дескриптора хранятся результаты последнего поиска. Каждый новый вызов функций из группы **`search`**… перезаписывает результаты предыдущего поиска по данному дескриптору.

Все указатели, получаемые в качестве выходных параметров функций необходимо использовать только для получения данных, хранимых в ресурсах библиотеки. Модификация по этим указателям может привести к порче структур данных, загруженных из библиотеки ФИАС и хранимых в оперативной памяти. Если это произошло, следует выгрузить экземпляр библиотеки по данному дескриптору и загрузить библиотеку снова.

Память под загруженные ресурсы освобождается функцией **`closeFiasAO`**.

## Перечень ресурсов библиотеки

Все пути указаны относительно корневой директории библиотеки, если библиотека поставляется в виде модуля для компоненты Адаптера FXServer.exe, то как правило это каталог «./FiasModule/».

Список ресурсов задаётся в конфигурационном файле, обычно «./config-fias.txt». Каждый ресурс задан в виде разделённых символом табуляции токенов строки, где на первом месте стоит путь к файлу, на втором – тип ресурса (см. таблицу), и в конце – формат («TX» или «GZ»).

Обычно сами файлы ресурсов располагаются в каталоге «./fias_xml/». Основные ресурсы представлены избранными архивированными файлами справочника адресов ФИАС с официального сайта. Вспомогательные передаются в виде текстовых файлов.

| **Название файла** | **Обозначение** | **Описание** |
|---|---|---|
| AS_ADDROBJ_date_GUID.XML.gzip | ADDROBJ | Иерархия основных (именованных) адресных элементов с кодами ФИАС |
| AS_HOUSE_date_GUID.XML.gzip | HOUSES | Список домов, подчинённых основным адресным элементам |
| AS_ROOM_date_GUID.XML.gzip | FLATS | Список квартир, подчинённых домам |
| AS_SOCRBASE_date_GUID.XML | SOCRBASE | Список основных сокращений элементов ФИАС с их уровнями в иерархии |
| rco_addr_socrs.tsv | SOCRSYN | Список возможных синонимов сокращений элементов |
| rco_addr_name_otch_gen.tsv | PERSNM | Имена и отчества для синонимов улиц |
| rco_addr_num_syns.tsv | NUMSYN | Синонимы номеров улиц |
| rco_addr_proffession.tsv | PROFSYN | Синонимы профессий |
| common_frags_ed.tsv | COMFRAG | Общая лексика, для исключения из однословных синонимов улиц |
| House_line_rules.txt | HLINEREGEX | Упорядоченный набор правил для разбора строки дома, корпуса и строения |
| Flat_line_rules.txt | FLINEREGEX | Упорядоченный набор правил для разбора строки квартира-помещение |

* В таблице на сером фоне перечислены данные, получаемые напрямую из выгрузки ФИАС с официального сайта, для обновления ФИАС требуется подменить только файлы с этими обозначениями в конфиге. В этих файлах «date» задана в формате «ггггммдд», «GUID» в формате «8-4-4-4-12» шестнадцатеричных разрядов. Жирным отмечены ресурсы, после загрузки которых из XML библиотека создаёт бинарные кэши для ускорения повторной загрузки.

# Описание функций библиотеки

## Функции инициализации и освобождения

### loadFiasAO

Функция загружает библиотеку загрузки и поиска по данным ФИАС в память с данными, описанными в конфигурационном файле. Если для много потоковой работы необходимо использовать несколько экземпляров библиотеки в одной и той же конфигурации, то первый экземпляр следует создавать с помощью этой функции, а последующие с помощью функции **`splitFiasHandle`**

```
int loadFiasAO(
   const char* pCfg, // путь к конфигурационному файлу
   bool bTryCache, // проверять ли кэш
   FiasHandle* pHandle //дескриптор ФИАС
);
```

**Параметры**
*pCfg*

[вх] Путь к конфигурационному файлу. В составе дистрибутива поставляется RCO_AP3_Service\FiasModule\config-fias.txt. Может быть сделан файл в таком формате с указанием других адресных элементов и баз для загрузки, и его имя и расположение передано в этом параметре.

*bTryCache*

[вх] Проверять ли при загрузке наличие ранее созданного бинарного кэша ресурсов, или же кэш будет проигнорирован, и библиотека загрузится из исходного XML дампа ФИАС, заново пересоздав кэш.

*pHandle*

[вых] Если библиотека успешно загрузила конфигурацию, то возвращается дескриптор для дальнейшей работы с библиотекой.

**Возвращаемое значение**
Если загрузка прошла успешно, функция возвращает **0**, в противном случае возвращается код ошибки.

## closeFiasAO

Функция освобождает память, занимаемую библиотекой загрузки и поиска по данным ФИАС, выделенную функциями **loadFiasAO** и **splitFiasHandle** и должна вызываться по завершению работы для дескрипторов, полученных этими функциями

```
bool closeFiasAO(
   FiasHandle handle // дескриптор библиотеки
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор библиотеки загрузки и поиска по данным ФИАС.

**Возвращаемое значение**
Функция возвращает **true**, если освобождение памяти прошло успешно и **false**, если не удалось.

## splitFiasHandle

Функция выделяет дополнительный дескриптор на загруженную библиотеку поиска по данным ФИАС для многопоточной работы. Загруженная конфигурация выгружается из

памяти, когда дескриптор, полученный вызовом **loadFiasAO**, и все его дополнительные дескрипторы освобождены функцией **closeFiasAO**.

Если нужна библиотека, сконфигурированная другим конфигурационным файлом, то следует использовать функцию **loadFiasAO**.

```
FiasHandle splitFiasHandle(
   FiasHandle handle //дескриптор загруженной библиотеки
);
```

**Параметры**
*handle*

[вх] Дескриптор библиотеки, возвращенный при вызове функции **loadFiasAO**.

**Возвращаемое значение**
Дескриптор для работы с библиотекой в той же конфигурации, что и передаваемый.

## Функции поиска по адресным элементам

Эти поисковые функции принимают список адресных полей в текстовом виде в кодировке Windows−1251. Полученные при поиске данные будут доступны по дескриптору библиотеки до последующего вызова одной из функций поиска по этому дескриптору.

## searchTypedNameSocrs

Функция осуществляет поиск адреса по адресным элементам, сокращениям и Cape−типам (см. Перечень поддерживаемых типов адресных элементов).

```
int searchTypedNameSocrs(
   FiasHandle handle, // дескриптор библиотеки
   const char** ppNames, // адресные элементы
   const char** ppSocrs, // сокращения
   const char** ppTypes, // Cape-типы
   size_t iPartCnt, // число имён, сокращений, Cape-типов
   bool bTryFuzzy // использовать ли нечеткое сравнение
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*ppNames*

[вх] Названия адресных элементов. Если элемент пропущен ставится пустое значение.

*ppSocrs*

[вх] Адресные сокращения. Если элемент пропущен ставится пустое значение.

*ppTypes*

[вх] Cape−типы элементов. Если элемент пропущен ставится пустое значение.

*iPartCnt*

[вх] Задает число имён/сокращений/Саре–типов. (Длина массивов ppNames, ppSocrs, ppTypes).

*bTryFuzzy*

[вх] Будет ли использоваться нечёткое сравнение строк. При этом в каждом элементе допускается одна опечатка (пропущенная, лишняя или изменённая буква), и максимум две опечатки во всех элементах суммарно.

Пример:
Адресная строка: Москва, ул. Вавилова, д.12

*ppNames*[] = {"Москва", "Вавилова", "12"};

*ppSocrs*[] = {"","ул","д"};

*ppTypes*[] = {"Address:Part_City", "Address:Part_Street", "Address:Part_House "};

*iPartCnt= 3*

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchNameSocrs

Функция осуществляет поиск адреса по адресным элементам, сокращениям.

```
int searchNameSocrs(
   FiasHandle handle, // дескриптор библиотеки
   const char** ppNames, // адресные элементы
   const char** ppSocrs, // сокращения
   size_t iPartCnt, // число имён, сокращений
   bool bTryFuzzy // использовать ли нечеткое сравнение
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*ppNames*

[вх] Названия адресных элементов.

*ppSocrs*

[вх] Адресные сокращения.

*iPartCnt*

[вх] Задает число имён/сокращений. (Длина массивов ppNames, ppSocrs).

*bTryFuzzy*

[вх] Будет ли использоваться нечёткое сравнение строк. При этом в каждом элементе допускается одна опечатка (пропущенная, лишняя или изменённая буква), и максимум две опечатки во всех элементах суммарно.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchTypedFields

Функция осуществляет поиск адреса по слитным названиям с сокращениями и Cape–типам.

```
int searchTypedFields(
   FiasHandle handle, // дескриптор библиотеки
   const char** ppNameSocrs, // адресные элементы
   const char** ppTypes, // Cape-типы
   size_t iPartCnt, //
   bool bTryFuzzy // использовать ли нечеткое сравнение
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*ppNameSocrs*

[вх] Названия адресных элементов с сокращением. Пример “Вавилова улица”

*ppTypes*

[вх] Cape–типы элементов

*iPartCnt*

[вх] Задает число имён/сокращений/Cape–типов. (Длина массивов ppNameSocrs, ppTypes).

*bTryFuzzy*

[вх] Будет ли использоваться нечёткое сравнение строк. При этом в каждом элементе допускается одна опечатка (пропущенная, лишняя или изменённая буква), и максимум две опечатки во всех элементах суммарно.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## Функции поиска по кодам и GUID ФИАС

## searchFiasCode

Функция позволяет быстро найти адресные элементы по известному коду ФИАС

```
int searchFiasCode(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iFiasCode // код ФИАС
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iFiasCode*

[вх] Иерархический код ФИАС адресного элемента.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchKladrCode

Функция поиска адресных элементов по коду КЛАДР

```
int searchKladrCode(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iKladrCode // код КЛАДР
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iKladrCode*

[вх] Код адресного элемента по КЛАДР.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchAOGUID

Функкция осуществляет поиск адресных объектов по GUID.

```
int searchAOGUID(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iGUID1, // старшие 8 байт GUID
   uint64_t iGUID2 // младшие 8 байт GUID
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iGUID1*

[вх] Старшие 8 байт целочисленного представления GUID адресного объекта.

*iGUID2*

[вх] Младшие 8 байт целочисленного представления GUID адресного объекта.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchHouseGUID

Функция осуществляет поиск домов по GUID.

```
int searchHouseGUID(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iGUID1, // старшие 8 байт GUID
   uint64_t iGUID2 // младшие 8 байт GUID
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iGUID1*

[вх] Старшие 8 байт целочисленного представления GUID дома.

*iGUID2*

[вх] Младшие 8 байт целочисленного представления GUID дома.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## searchAnyGUID

Функция самостоятельно пытается определить, какой GUID был передан: адресного объекта или дома, и вызвать соответствующий поиск (**`searchAOGUID`** или **`searchHouseGUID`**).

```
int searchAnyGUID(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iGUID1, // старшие 8 байт GUID
   uint64_t iGUID2 // младшие 8 байт GUID
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iGUID1*

[вх] Старшие 8 байт целочисленного представления GUID.

*iGUID2*

[вх] Младшие 8 байт целочисленного представления GUID.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## Функции получения результатов поиска.

Если функция поиска вернула значение больше нуля, это означает, что в библиотеке для использовавшегося дескриптора сохранены результаты поиска, и их можно получить с помощью описанных в этом разделе функций (до момента следующего вызова одной из функций поиска по этому дескриптору).

## getVarData

Функция позволяет получить нормализованную строку адреса, код ФИАС, уровень адресного элемента в иерархии, варианты домов, до десяти вероятных вариантов-подсказок продолжения для введенного адреса и старый код КЛАДР. Передаваемым выходным параметрам указателям присваиваются адреса соответствующих данных, размещенных по заданному дескриптору в библиотеке. И данные, получаемые по ним, следует использовать только для чтения.

```
int getVarData(
   FiasHandle handle, // дескриптор библиотеки
   size_t iVarIndex, // индекс варианта разбора
   const char** ppFullName, // нормализованная строка адреса
   uint64_t* pFiasCode, // код ФИАС
   unsigned int* pLvl, // уровень адресного элемента в иерархии
   int* pHouseMatchRes, // число вариантов домов
   int* pContinueVars, // число вариантов задания строки адреса
   uint64_t* pKladrCode //код КЛАДР
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*ppFullName*

[вых] Указатель, на нормализованную строку адреса.

*pFiasCode*

[вых] Указатель на иерархический код ФИАС.

*pLvl*

[вых] Указатель на уровень адресного элемента в иерархии.

*pHouseMatchRes*

[вых] Указатель на число вариантов домов для этого варианта разбора.

*pContinueVars*

[вых] Указатель на число вероятных вариантов задания строки продолжения адреса (до десяти штук).

*pKladrCode*

[вых] Указатель на код в старой адресной базе КЛАДР.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## getVarHouseData

Функция позволяет получить почтовый индекс, GUID, и строки с домом, корпусом и квартирой, для каждого подходящего дома, используя число вариантов домов `pHouseMatchRes` (для каждого из вариантов разбора, полученных функцией **`getVarData)`**.

```
int getVarHouseData(
   FiasHandle handle, // дескриптор библиотеки
   size_t iVarIndex, // индекс варианта разбора
   size_t iHouseIndex, // индекс варианта дома
   unsigned int* iPostIndex, // Указатель для почтового индекса
   uint64_t* pGUID1, // указатель для получения старших 8 байт GUID
   uint64_t* pGUID2, // указатель для получения младших 8 байт GUID
   const char** ppHouseLine, //указатель на строку с домом и корпусом
   const char** ppFlatLine // указатель на строку с квартирой
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*iHouseIndex*

[вх] Индекс запрашиваемого варианта дома(индексы идут с нуля).

*iPostIndex*

[вых] Указатель для получения почтового индекса.

*pGUID1*

[вых] Указатель для получения старших 8 байт целочисленного представления GUID адресного объекта. (AO)

*pGUID2*

[вых] Указатель для получения младших 8 байт целочисленного представления GUID адресного объекта. (AO)

*ppHouseLine*

[вых] Указатель на строку для получения дома с корпусом.

*ppFlatLine*

[вых] Указатель на строку для получения квартиры.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

**Пример**
Пусть у нас есть адрес, разбор которого привел к вариантам домов в формате JSON:


```
"house_variants": [
          {
            "house_line": "ДОМ 4",
            "post_index": 143900,
            "HOUSE_GUID": "f1fff92f-54b9-4ec2-8567-d9e68a5da6d2",
            "appartment": "КВ 59",
            "appartment_matched": true
          }
        ],
```


Для того, чтобы их разобрать воспользуемся функцией **`getVarHouseData`**

```
const char* pHouseLine = 0;
const char* pFlatLine = 0;
FiasResVarSimple::HouseFlatMatch hf;
int iVarResHouse = getVarHouseData(
   g_FiasHandle, i, j, &hf.iPostIndex, &hf.iHouseGUID1,
   &hf.iHouseGUID2, &pHouseLine, &pFlatLine, &hf.bFlatMatched );
```

## getVarContinue

Функция позволяет получить строки вариантов продолжения адресной строки. Эту функцию можно использовать для подсказки в строке поиска.

```
int getVarContinue(
   FiasHandle handle, // дескриптор библиотеки
```

```
   size_t iVarIndex, // индекс варианта разбора
   size_t iContinueIndex, // индекс варианта продолжения
   const char** ppContinueLine//указатель адресной строки продолжения
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*iContinueIndex*

[вх] Индекс запрашиваемого варианта продолжения адресной строки (индексы идут с нуля).

*ppContinueLine*

[вых] Указатель для получения текста варианта продолжения адресной строки.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## getVarDataAux

Функция позволяет получить для выбранного варианта разбора различную дополнительную информацию: GUID адресного элемента, коды ОКАТО/ОКТМО/почтовый индекс (если эти данные для адресного элемента не заданы, то соответствующим выходным параметрам функции присваивается нуль).

```
int getVarDataAux(
   FiasHandle handle, // дескриптор библиотеки
   size_t iVarIndex, // индекс варианта разбора
   uint64_t* pGUID1, // указатель для получения старших 8 байт GUID
   uint64_t* pGUID2, // указатель для получения младших 8 байт GUID
   uint64_t* pOKATO, // указатель для получения кода ОКАТО
   uint64_t* pOKTMO, // указатель для получения кода ОКТМО
   uint32_t* pPostalIndex // указатель для почтового индекса
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*pGUID1*

[вых] Указатель для получения старших 8 байт целочисленного представления GUID адресного объекта. (АО)

*pGUID2*

[вых] Указатель для получения младших 8 байт целочисленного представления GUID адресного объекта. (АО)

*pOKATO*

[вых] Указатель для получения кода ОКАТО выбранного варианта разбора.

*pOKTMO*

[вых] Указатель для получения кода ОКТМО выбранного варианта разбора.

*pPostalIndex*

[вых] Указатель для получения почтового индекса выбранного варианта разбора.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## getVarPartMatches

Функция позволяет определить позиции исходных (поданных на поиск) частей адреса в результирующей нормализованной строке.

```
int getVarPartMatches(
   FiasHandle handle, // дескриптор библиотеки
   size_t iVarIndex, // индекс варианта разбора
   unsigned char* pAOPartMatches, // указатель на индексы частей
   unsigned char* pMatchMode, // указатель на тип соответствия частей
   size_t* ppPartMatchesCount // указ. на размеры буферов для частей
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*pAOPartMatches*

[вх вых] Буфер для индексов, указывающих порядок следования адресного элемента в составленной нормализованной строке (память выделяется в клиентском приложении). Если для какой-то позиции нормализованной строки нет соответствия в исходных частях адреса, то в буфер записывается -1 (255).

*pMatchMode*

[вх вых] Буфер типа соответствия. В буфер записывается нуль при нормальном соответствии и другое значение при нечётком соответствии (с исправлением опечаток). Примечание: объявлены несколько типов соответствий:

```
enum eResMatchType {
   eRMT_Strict = 0, // строгое соответствие
   eRMT_History, // 1 = совпадение по истории наименования
   eRMT_Fuzzy, // 2 = соответствие полученное в ходе нечеткого поиска
};
```

*ppPartMatchesCount*

[вх вых] Размер доступных для заполнения буферов частей, (16 частей достаточно для любого адреса). В ходе работы функция изменяет размер, переданный по этому указателю, на число реально записанных значений.

**Пример использования**

```
varSimple.vMatchedParts.resize( 16 );
varSimple.vMatchMode.resize( 16 );
size_t iPartMatchesCount = varSimple.vMatchedParts.size();
int iMatchRes = getVarPartMatches(
   g_FiasHandle,
   i,
   &varSimple.vMatchedParts[0],
   &varSimple.vMatchMode[0],
   &iPartMatchesCount
);
varSimple.vMatchedParts.resize( iPartMatchesCount );
```

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## getVarPartInfo

Функция позволяет получить расширенную информацию для заданной части адреса: записанные в ФИАС текстовые поля названия и сокращения адресного элемента, его уровень в иерархии ФИАС, а также код ФИАС этого элемента.

```
int getVarPartInfo(
   FiasHandle handle, // дескриптор библиотеки
   size_t iVarIndex, // индекс варианта разбора
   size_t iPartIndex, // индекс части адреса
   const char** pFiasPartName, //ук. на название адресного элемента
   const char** pFiasPartSocr, //ук. на сокращение адресного элемента
   int* pFiasPartLvl, // указатель на уровень в иерархии ФИАС
   uint64_t* pFiasPartCode // указатель для получения кода ФИАС
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iVarIndex*

[вх] Индекс запрашиваемого варианта разбора (индексы идут с нуля).

*iPartIndex*

[вх] Индекс запрашиваемой части адреса (индексы идут с нуля).

*pFiasPartName*

[вх вых] Указатель для получения названия адресного элемента.

*pPartSocr*

[вых] Указатель для получения сокращения адресного элемента.

*pFiasPartLvl*

[вых] Указатель для получения уровня иерархии ФИАС.

*pFiasPartCode*

[вх вых] Указатель для получения кода ФИАС выбранного элемента.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество вариантов разбора адреса. В случае ошибки возвращается значение меньше нуля.

## getNormQueryParts

Функция производит нормализацию части адреса по тем же правилам, что и строки вариантов. Это может быть полезно, если адрес не был найден (нет вариантов разбора), тогда строка, составленная из нормализаций частей адреса, может хоть как-то помочь при идентификации. Получаемое по указателю значение параметра `ppNormQueryLine` соответствует `norm_parts_line` в ответе JSON.

```
int getNormQueryParts(
   FiasHandle handle, // дескриптор библиотеки
   const char** ppNormQueryLine, //нормализованные части адреса
   size_t iPartIndex, // индекс нормализуемой части
   const char** ppName, // указатель на нормализованное название
   const char** ppSocr // указатель на нормализованное сокращение
   const char** ppType, // указатель на Саре-тип заданной части
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*ppNormQueryLine*

[вх вых] Указатель, в который будет записан указатель на строку из нормализованных частей.

*iPartIndex*

[вх] Индекс запрашиваемой части адреса (индексы идут с нуля), для которой мы хотим получить нормализацию.

*ppName*

[вх вых] Указатель на название адресного элементам, полученного в результате нормализации заданной части. Необязательный параметр, может быть `Null`, если нет необходимости получать нормализацию заданной `iPartIndex` части.

*ppSocr*

[вх вых] Указатель на сокращение элемента, полученного в результате нормализации заданной части. Необязательный параметр, может быть `Null`, если нет необходимости получать нормализацию заданной `iPartIndex` части.

*ppType*

[вх вых] Указатель на Cape–тип элемента, полученного в результате нормализации заданной части. Необязательный параметр, может быть `Null`, если нет необходимости получать нормализацию заданной `iPartIndex` части.

**Возвращаемое значение**
В случае успешного завершения функция возвращает количество нормализованных частей. В случае ошибки возвращается значение меньше нуля (если переданный дескриптор не валиден или же `iPartIndex` больше количества нормализованных частей).

## Вспомогательные функции

Библиотека также предоставляет несколько полезных функций для работы с различными кодами, содержащимися в ФИАС.

### compareCodes

Функция позволяет сравнить два иерархических кода ФИАС.

```
int compareCodes(
   uint64_t codeA, // код ФИАС
   uint64_t codeB // код ФИАС
);
```

**Параметры**
*codeA*

[вх] Иерархический код ФИАС.

*codeB*

[вх] Иерархический код ФИАС.

**Возвращаемое значение**
Функция возвращает:

- нуль, если A и B никак не пересекаются;
- значение меньше нуля, если A соответствует одному из более общих уровней иерархии кода B;
- значение больше нуля, если коды либо равны, либо A является уточнением B более глубокой вложенности.

## dumpGUIDCanonicaly

Функция переводит целочисленное представление GUID в строковое. Каноническим считается представление GUID в виде строки «8-4-4-4-12» шестнадцатеричных разрядов.

```
void dumpGUIDCanonicaly(
   uint64_t iG1, // целочисленное представление: старшие 8 байт
   uint64_t iG2, // целочисленное представление: младшие 8 байт
   char* pBuff, // буфер для строкового представления
   unsigned int iBuffLen //длина буфера
);
```

**Параметры**
*iG1*

[вх] Старшие 8 байт целочисленного представления GUID для преобразования

*iG2*

[вх] Младшие 8 байт целочисленного представления GUID для преобразования

*pBuff*

[вх вых] Буфер, который будет заполнен строковым представлением GUID (память выделяется в прикладной программе, библиотека ее использует для записи).

*iBuffLen*

[вх] Длина буффера

**Возвращаемое значение**
Функция не возвращает значение.

## setGUIDFromString

Функция переводит строковое представление GUID в целочисленное. Каноническим считается представление GUID в виде строки «8-4-4-4-12» шестнадцатеричных разрядов.

```
bool setGUIDFromString(
   const char* pGUID, // строка с GUID
   size_t iLen, // размер строки с GUID
   uint64_t* pG1, // указатель для получения младших байт GUID
   uint64_t* pG2 // указатель для получения старших байт GUID
```

```
);
```

**Параметры**
*pGUID*

[вх] Строковое представление GUID для преобразования

*iLen*

[вх] Длина строкового представления

*pG1*

[вх вых] Указатель для получения старших 8 байт целочисленного представления GUID.

*iG2*

[вх вых] Указатель для получения младших 8 байт целочисленного представления GUID.

**Возвращаемое значение**
Функция возвращает true, если преобразование прошло успешно и false в случае ошибки.

## getFiasCodeLvl

Функция округленно вычисляет уровень иерархии кода ФИАС. Точный уровень указан в базе и может быть получен только функцией поиска по коду.

```
unsigned short getFiasCodeLvl(
   uint64_t iFiasCode // код ФИАС
);
```

**Параметры**
*iFiasCode*

[вх] код ФИАС

**Возвращаемое значение**
Функция возвращает округленный уровень иерархии кода ФИАС.

## getKladrCodeLvl

Функция вычисляет уровень иерархии кода КЛАДР.

```
unsigned short getKladrCodeLvl(
   uint64_t iKladrCode // код КЛАДР
);
```

**Параметры**
*iKladrCode*

[вх] Код КЛАДР

**Возвращаемое значение**
Функция возвращает уровень иерархии кода КЛАДР.

## getNameSocrForCode

Функция предназначена для нахождения значения самого верхнего элемента ФИАС, соответствующего переданному коду.

```
int getNameSocrForCode(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iFiasCode, // код ФИАС
   char* pNameBuff, // Буфер для наименования
   size_t iBuffSize, // размер буфера для наименования
   uint64_t* iGUID1, // указатель для получения старших 8 байт GUID
   uint64_t* iGUID2, // указатель для получения младших 8 байт GUID
   uint64_t* iOKATO, // указатель для получения кода ОКАТО
   uint64_t* iOKTMO // указатель для получения кода ОКТМО
);
```

**Параметры**

*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iFiasCode*

[вх] Код ФИАС, для которого ищется верхний элемент.

*pNameBuff*

[вх вых] Буфер для наименования верхнего элемента ФИАС.

*iBuffSize*

[вх] Размер буфера для наименования верхнего элемента ФИАС.

*iGUID1*

[вх вых] Указатель для получения старших 8 байт целочисленного представления GUID верхнего элемента ФИАС.

*iGUID2*

[вх вых] Указатель для получения младших 8 байт целочисленного представления GUID верхнего элемента ФИАС.

*iOKATO*

[вх вых] Указатель для получения кода ОКАТО верхнего элемента ФИАС.

*iOKTMO*

[вх вых] Указатель для получения кода ОКТМО верхнего элемента ФИАС.

**Возвращаемое значение**

Функция возвращает количество найденных вариантов элемента.

## getNameSocrForOKTMO

Функция предназначена для нахождения значения самого верхнего элемента ФИАС, соответствующего переданному коду ОКТМО.

```
int getNameSocrForOKTMO(
   FiasHandle handle, // дескриптор библиотеки
   uint64_t iCodeOKTMO, // код ОКТМО для поиска
   char* pNameBuff, // Буфер для наименования
   size_t iBuffSize, // размер буфера для наименования
   uint64_t* iGUID1, // указатель для получения старших 8 байт GUID
   uint64_t* iGUID2, // указатель для получения младших 8 байт GUID
   uint64_t* iOKATO, // указатель для получения кода ОКАТО
   uint64_t* iOKTMO // указатель для получения кода ОКТМО
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*iCodeOKTMO*

[вх] Код ОКТМО, для которого ищется верхний элемент.

*pNameBuff*

[вх вых] Буфер для наименования верхнего элемента ФИАС.

*iBuffSize*

[вх] Размер буфера для наименования верхнего элемента ФИАС.

*iGUID1*

[вх вых] Указатель для получения старших 8 байт целочисленного представления GUID верхнего элемента ФИАС.

*iGUID2*

[вх вых] Указатель для получения младших 8 байт целочисленного представления GUID верхнего элемента ФИАС.

*iOKATO*

[вх вых] Указатель для получения кода ОКАТО верхнего элемента ФИАС.

*iOKTMO*

[вх вых] Указатель для получения кода ОКТМО верхнего элемента ФИАС.

**Возвращаемое значение**
Функция возвращает количество найденных вариантов элемента.

## getNameSocrForOKTMO

Функция предназначена для нормализации сокращений.

```
int normSocr(
   FiasHandle handle, // дескриптор библиотеки
   const char* pType, // Cape-тип элемента
   const char* pSocr, // сокращение
   const char** ppNormSocr // указ. на нормализованное сокращение
);
```

**Параметры**
*FiasHandle*

[вх] Дескриптор инициализированной библиотеки загрузки и поиска по данным ФИАС.

*pType*

[вх] Cape–тип элемента, сокращение которого требуется нормализовать.

*pSocr*

[вх] Сокращение, которое требуется нормализовать.

*ppNormSocr*

[вх вых] Указатель будет задан адресом памяти, выделенной библиотекой, где находится нормализованное сокращение.

**Возвращаемое значение**
Функция возвращает число вариантов нормализации сокращения.

## Список поддерживаемых типов адресных элементов

Библиотека поддерживает следующие типы адресных элементов:

```
"Address:Part_Country", "Address:Part_Subject", "Address:Part_Region",
"Address:Part_City", "Address:Part_Settlement", "Address:Part_District",
"Address:Part_Street", "Address:Part_House", "Address:Part_Building",
"Address:Part_SubHouse", "Address:Part_Floor", "Address:Part_Apartment",
"Address:Part_Room", "Address:Part_Index".
```

В описании интерфейса для обозначения типа адресного элемента часто используется термин “Cape-тип”, отсылающий к модулю CAPE (C++ Annotation Patterns Engine), входящему в состав RCOFxRux64.dll, служащему для извлечения упоминаний сложных объектов из текста на основе правил.

# Программный интерфейс для СУБД Oracle

## Общее описание

Интерфейс предназначен для вызова функций RCO АРЗ программным кодом PL/SQL под управлением СУБД Oracle (процедурами и пакетами различных схем).

Служба разбора адресов не требует для своей работы СУБД Oracle. Данный интерфейс является дополнительным инструментарием для обращения к Службе разбора адресов.

Интерфейс реализован в виде схемы пользователя с наименованием RCO_AP3 для СУБД Oracle версии 12c или выше.

Схема состоит из открытых пакетов, интерфейсных типов входных данных и данных результата API, а также вспомогательных пакетов, таблиц в табличном пространстве USERS, или собственном табличном пространстве, например, RCO_ADDR_TBS, если оно было специально задано при выполнении скрипта установки (Подробнее см. Руководство Администратора, раздел Установка пакетов для Oracle)

Установка и первоначальная настройка пакета выполняется скриптами Oracle администратором СУБД.

Доступ к открытым пакетам или типам данных должен быть предоставлен целевым схемам стандартным оператором Oracle “grant execute on <тип или пакет> to <схема>;”.

Основная функция интерфейса − осуществлять запросы к Службе разбора адресов и разбирать возвращаемую информацию.

## Функции разбора и нормализации адресов

### Пакет PKG_FX_ADDR

Пакет PKG_FX_ADDR - основной пакет API для запроса анализа адресной информации.

Инкапсулирует протокол обращения к Адаптеру АРЗ Службы разбора адресов, позволяет оперировать с входной информацией и данными результата в виде структур типов Oracle.

С целью уменьшения накладных расходов по обращению к внешнему сервису по протоколу HTTP, в одном запросе позволено задать множество почтовых адресов для обработки.

Для обеспечения возможности проведения HTTP запроса, необходимо предоставить пользователю RCO_AP3 соответствующие права Oracle ACL доступа к URL Адаптера (к адресу с порт, который прослушивает компонента Адаптера FXServer.exe).

```
create or replace package pkg_fx_addr
as

-- установить текущий URL FX сервера сеанса
procedure set_fx_url
(
  p_url varchar2 := null -- FX server URL, null -> default =
   pkg_Settings.get_setting_value( 'URL_DEFAULT' )
);

-- получить текущий URL FX сервера сеанса
function get_fx_url return varchar2;

--
-- Методы анализа адресов
--

-- обработка адресов, возвращает JSON
function fx_get_addr_json
(
  p_addr_in R_RCO_ADDR_IN -- структура массива адресов
)
return clob; -- JSON результат обработки

-- обработка адресов, возвращает T_RCO_ADDR_OUT
function fx_get_addr
(
  p_addr_in R_RCO_ADDR_IN -- структура массива адресов
)
return T_RCO_ADDR_OUT; -- массив структур выходных данных результата
   обработки адресов

end pkg_fx_addr;
```

Пакет позволяет:

- Установить и получить URL сервера SOAP Службы разбора адресов для текущего сеанса.

  URL по умолчанию хранится в таблице настроек, задаётся при установке системы.
- Обработать входной массив запросов адресной информации разных видов, заданных переменной типа R_RCO_ADDR_IN, и вернуть для каждого адреса результат или ошибку в массиве структур T_RCO_ADDR_OUT.

**set_fx_url**

Процедура устанавливает значение адреса, по которому доступен Адаптер AP3.

```
procedure set_fx_url(
   p_url varchar2 := null -- [url:порт] Адаптера AP3
```

```
);
```

**Параметры**

*p_url*

[вх] Устанавливает значение адреса и порт Адаптера АР3 в формате <адрес>:<порт> (см. Руководство администратора раздел 4.2.1 п.2 настройку файла RCO_AP3_Service\ap3_service.cmd для запуска FXServer.exe). Значение по умолчанию параметра `Null`. Оно соответствует значению, настройки URL_DEFAUL, которую можно получить обращением к пакету настроек **`pkg_Settings.get_setting_value('URL_DEFAULT')`**.

**set_fx_url**

Процедура устанавливает значение адреса, по которому будут отправлены запросы к Адаптеру АР3. Процедура может вызываться, когда необходимо изменить значение, сохраненное при установке пакетов.

```
procedure set_fx_url(
   p_url varchar2 := null -- [url:порт] Адаптера АР3
);
```

**Параметры**

*p_url*

[вх] Значение адреса и порта Адаптера АР3 в формате <адрес>:<порт> (см. Руководство администратора раздел 4.2.1 п.2 настройку файла RCO_AP3_Service\ap3_service.cmd для запуска FXServer.exe). Значение по умолчанию параметра `Null`. Оно соответствует значению настройки `URL_DEFAULT`, которую можно получить обращением к пакету настроек **`pkg_Settings.get_setting_value('URL_DEFAULT')`**.

**get_fx_url**

Функция позволяет получить значение адреса и порта настройки на Адаптер АР3.

```
function get_fx_url return varchar2;
```

**Возвращаемое значение**

Функция возвращает строку в формате <адрес>:<порт>, которая будет использована для подключения и отправки адресного запроса к Адаптеру АР3. См. также **`set_fx_url`**.

**fx_get_addr_json**

Функция производит обработку адресов, заданных во входном объекте типа **`R_RCO_ADDR_IN`** и возвращает результаты разбора в формате `json`. Функция не

предназначена для запуска пользователем. Для разбора адресов следует использовать fx_get_addr, которая возвращает структурированный ответ.

```
function fx_get_addr_json(
  p_addr_in R_RCO_ADDR_IN -- структура массива адресов
)
 return clob; -- JSON результат обработки
```

**Параметры**

*p_addr_in*

[вх] Входные адреса для разбора в виде объекта типа **R_RCO_ADDR_IN**.

**Возвращаемое значение**

Функция возвращает результаты разбора адресов в виде clob в формате json.

### fx_get_addr

Функция производит обработку адресов, заданных во входном объекте типа R_RCO_ADDR_IN и возвращает результаты разбора в выходном объекте типа T_RCO_ADDR_OUT, являющимся таблицей объектов типа R_RCO_ADDR_OUT. Описание объектов входных и выходных данных описаны ниже. Функция является основной для разбора адресов. См. также пример обработки адреса.

```
function fx_get_addr(
  p_addr_in R_RCO_ADDR_IN -- структура массива входных адресов
)
 return T_RCO_ADDR_OUT; -- массив структур выходных данных результата
```

**Параметры**

*p_addr_in*

[вх] Входные адреса для разбора в виде объекта типа **R_RCO_ADDR_IN**.

**Возвращаемое значение**

Функция возвращает результаты разбора адресов в виде объекта табличного типа **T_RCO_ADDR_OUT**.

## Объектный тип входных адресных данных R_RCO_ADDR_IN

Используется в качестве входного параметра для функций поиска, обращающихся к Службе разбора адресов. Объект данного типа хранит множество адресных запросов.

```
CREATE OR REPLACE TYPE "R_RCO_ADDR_IN"
--
-- структура входных данных
--
as object(
  addr_cnt integer, -- количество адресов
  addr_json json, -- множество адресов в формате json
--
-- конструкторы
```

```
--

   CONSTRUCTOR FUNCTION R_RCO_ADDR_IN RETURN SELF AS RESULT,

--
-- методы
--

  member procedure addrAdd(
   p_addr_id varchar2,
   p_addr_json json
  ),

  -- добавить полный адрес в структуру адресов
  member procedure addrAddFull(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_is_birth_address boolean := false, --флаг "адрес-место рождения"
   p_try_fuzzy boolean := false, --флаг "использовать нечёткий поиск"
   p_address_line VARCHAR2 -- полная строка адреса
  ),

  -- добавить адрес по полям в структуру адресов
  member procedure addrAddParts(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_is_birth_address boolean := false, --флаг "адрес-место рождения"
   p_try_fuzzy boolean := false, --флаг "использовать нечёткий поиск"
   p_Part_Country VARCHAR2 := null, -- страна
   p_Part_Subject VARCHAR2 := null, -- субъект
   p_Part_Region VARCHAR2 := null, -- регион
   p_Part_City VARCHAR2 := null, -- город
   p_Part_District VARCHAR2 := null, -- район города
   p_Part_Street VARCHAR2 := null, -- улица
   p_house_line VARCHAR2 := null, -- дом
   p_flat_line VARCHAR2 := null, -- квартира
   p_Part_Index VARCHAR2 := null -- индекс
  ),

  -- добавить ФИАС GUID в структуру адресов
  member procedure addrAddFiasGuid(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_fias_guid VARCHAR2 -- ФИАС GUID
  ),

  -- добавить ФИАС код в структуру адресов
  member procedure addrAddFiasCode(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_fias_code NUMBER -- ФИАС код
  ),

  -- добавить КЛАДР код в структуру адресов
  member procedure addrAddKladrCode(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_kladr_code NUMBER -- КЛАДР код
  )

);
```

Содержит поля для хранения данных входных запросов и методы добавления к этим данным запросов адресной информации различного вида.

*addr_cnt*

Целое поле, характеризующее количество адресов, которое храниться в структуре.

*addr_json*

Поле типа JSON, хранящее адреса структуры.

### CONSTRUCTOR FUNCTION R_RCO_ADDR_IN

Конструктор без параметров, позволяющий создать экземпляр пустого объектного типа **`r_rco_addr_in`**. Производит инициализацию полей.

### addrAdd

Вспомогательная процедура, добавляет JSON в объект. Не предназначена для вызова прикладным программистом, для добавления адресных запросов следует использовать другие процедуры `addrAddxxx`.

```
member procedure addrAdd(
 p_addr_id varchar2, -- необязательный идентификатор адреса
 p_addr_json json -- json
)
```

**Параметры**
*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_addr_json*

[вх] добавляемый JSON.

### addrAddFull

Добавляет запрос по неструктурированному полному адресу.

```
member procedure addrAddFull(
 p_addr_id varchar2 := null, -- необязательный идентификатор адреса
 p_is_birth_address boolean := false, --флаг "адрес-место рождения"
 p_try_fuzzy boolean := false, --флаг "использовать нечёткий поиск"
 p_address_line VARCHAR2 -- полная строка адреса
)
```

**Параметры**
*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_is_birth_address*

[вх] Флаг, является ли адрес местом рождения: `false`, если является и `true`, если нет. По умолчанию адрес не считается местом рождения.

*p_try_fuzzy*

[вх] Флаг, использовать ли нечеткий поиск при разборе адреса. По умолчанию нечеткий поиск не используется.

*p_address_line*

[вх] Полная строка адресного запроса.

### addrAddParts

Процедура добавляет структурированный запрос по полям почтового адреса.

```
member procedure addrAddParts(
 p_addr_id varchar2 := null, -- необязательный идентификатор адреса
 p_is_birth_address boolean := false, --флаг "адрес-место рождения"
 p_try_fuzzy boolean := false, --флаг "использовать нечёткий поиск"
 p_Part_Country VARCHAR2 := null, -- страна
 p_Part_Subject VARCHAR2 := null, -- субъект
 p_Part_Region VARCHAR2 := null, -- регион
 p_Part_City VARCHAR2 := null, -- город
 p_Part_District VARCHAR2 := null, -- район города
 p_Part_Street VARCHAR2 := null, -- улица
 p_house_line VARCHAR2 := null, -- дом
 p_flat_line VARCHAR2 := null, -- квартира
 p_Part_Index VARCHAR2 := null -- индекс
)
```

**Параметры**

*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_is_birth_address*

[вх] Флаг, является ли адрес местом рождения: `false`, если является и `true`, если нет. По умолчанию адрес не считается местом рождения.

*p_try_fuzzy*

[вх] Флаг, использовать ли нечеткий поиск при разборе адреса. По умолчанию нечеткий поиск не используется.

*p_Part_Country*

[вх] Поле адреса: страна.

*p_Part_Subject*

[вх] Поле адреса: субьект.

*p_Part_Region*

[вх] Поле адреса: регион.

*p_Part_City*

[вх] Поле адреса: город.

*p_Part_District*

[вх] Поле адреса: район города.

*p_Part_Street*

[вх] Поле адреса: улица.

*p_house_line*

[вх] Поле адреса: дом.

*p_flat_line*

[вх] Поле адреса: квартира.

*p_Part_Index*

[вх] Поле адреса: индекс.

### addrAddFiasGuid

Процедура добавляет запрос по GUID базы ФИАС.

```
member procedure addrAddFiasGuid(
 p_addr_id varchar2 := null, -- необязательный идентификатор адреса
 p_fias_guid VARCHAR2 -- ФИАС GUID
)
```

**Параметры**
*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_fias_guid*

[вх] код GUID в базе ФИАС

### addrAddFiasCode

Процедура добавляет запрос по коду ФИАС.

```
member procedure addrAddFiasCode(
 p_addr_id varchar2 := null, -- необязательный идентификатор адреса
 p_fias_code NUMBER -- ФИАС код
)
```

**Параметры**
*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_fias_code*

[вх] код ФИАС

**addrAddKladrCode**

Процедура добавляет запрос по коду КЛАДР.

```
  member procedure addrAddKladrCode(
   p_addr_id varchar2 := null, -- необязательный идентификатор адреса
   p_kladr_code NUMBER -- КЛАДР код
  )
```

**Параметры**
*p_addr_id*

[вх] Необязательный идентификатор адреса.

*p_fias_code*

[вх] код КЛАДР

## Типы элемента структуры возвращаемого результата разбора адреса R_RCO_ADDR_OUT

Каждый отдельный результат разбора адреса хранится в структуре типа данных R_RCO_ADDR_OUT в порядке, соответствующем конкретному входному запросу.

```
CREATE OR REPLACE TYPE "R_RCO_ADDR_OUT"
--
-- структура выходных данных результата обработки адреса
--
as object(
  error VARCHAR2(1000 CHAR), -- сообщение об ошибке запроса поиска
   адреса
  address_id VARCHAR2(100 CHAR),--необязательный идентификатор адреса
  norm_parts_line VARCHAR2(1000 CHAR),--нормализованная строка адреса
  variants T_FIAS_VARIANT, -- варианты поиска в ФИАС
  fields T_FIELD_PART, -- поля адреса
  fields_norm T_FIELD_PART, -- нормализованные поля адреса
  russian number(1), -- 0 если адрес не русский, иначе 1
  USSR number(1), -- 0 если это не Россия _и_ не СССР, иначе 1
  OKSM number(3) -- код страны, 0 - Россия
);
```

*error*

Поле со строкой сообщения об ошибке, если таковая произошла, и `Null`, если разбор адреса прошел без ошибок.

*address_id*

Поле, хранящее необязательный идентификатор адреса.

*norm_parts_line*

Поле с нормализованной строкой адреса.

*variants*

Поле, хранящее массив вариантов поиска в ФИАС.

*fields*

Поле, хранящее массив элементов структурированного адреса.

*fields_norm*

Поле, хранящее массив нормализованных элементов структурированного адреса.

*russian*

Целочисленное поле-флаг, принимающее значение 1 при принадлежности адреса РФ, и 0 в остальных случаях (иностранные государства, СССР).

*USSR*

Целочисленное поле-флаг, принимающее значение 1 при принадлежности адреса к территории СССР, и 0 в остальных случаях (иностранные государства – не РФ и не СССР).

*OKSM*

Целочисленное поле, хранящее код страны по классификатору ОКСМ.

## Тип таблицы структур возвращаемого результата разбора адреса T_RCO_ADDR_OUT

Результаты разбора адреса возвращаются виде таблицы адресов.

```
CREATE OR REPLACE TYPE "T_RCO_ADDR_OUT" is table of R_RCO_ADDR_OUT;
```

## Тип элемента структурированного адреса R_FIELD_PART

Тип содержит поля для хранения адресного элемента, сокращения и его Саре–типа (см. Список поддерживаемых адресных элементов).

```
CREATE OR REPLACE TYPE "R_FIELD_PART"
--
-- структура адресных полей
--
as object(
  name VARCHAR2(100 CHAR), --значение ("МОСКОВСКАЯ","ПУШКИНСКИЙ", ..)
  key VARCHAR2(50 CHAR), -- ключ ("ОБЛ", "Р-Н", "Г", ..)
  type VARCHAR2(50 CHAR) -- тип поля ("Address:Part_Subject",
   "Address:Part_Region", "Address:Part_City", ..)
);
```

*name*

Название адресного элемента.

*key*

Сокращение адресного элемента.

*type*

Саре–тип адресного элемента.

### Тип адресных элементов структурированного адреса T_FIELD_PART

Объект позволяет хранить адрес структурированно в виде массива структурированных адресных элементов.

```
CREATE OR REPLACE TYPE "T_FIELD_PART" is table of R_FIELD_PART;
```

### Тип варианта записи о доме R_HOUSE_VARIANT

Содержит информацию о доме

```
CREATE OR REPLACE TYPE "R_HOUSE_VARIANT"
--
-- структура информации о доме
--
as object(
  house_line VARCHAR2(100 CHAR), -- номер дома
  post_index number(20), -- почтовый индекс
  appartment VARCHAR2(100 CHAR), -- помещение
  HOUSE_GUID VARCHAR2(36 CHAR) -- GUID строения
);
```

*house_line*

Строка с номером дома.

*post_index*

Строка с почтовым индексом.

*appartment*

Строка с номером помещения.

*HOUSE_GUID*

GUID строения в базе ФИАС.

### Тип таблицы вариантов записей о домах T_HOUSE_VARIANT

Является таблицей, содержащей информацию о возможных вариантах домов.

```
CREATE OR REPLACE TYPE "T_HOUSE_VARIANT" is table of R_HOUSE_VARIANT;
```

### Тип варианта поиска в ФИАС R_FIAS_VARIANT

Объекты данного типа предназначены для хранения найденного варианта адреса, соответствующего адресному запросу.

```
CREATE OR REPLACE TYPE "R_FIAS_VARIANT"
--
-- структура соответствия записи ФИАС
--
as object(
  norm_line VARCHAR2(1000 CHAR), -- нормализованная строка адреса
  norm_line_lev_dist number(3), -- дистанция нормализованного адреса
  FIAS_GUID VARCHAR2(50 CHAR), -- GUID ФИАС (неизменен)
  FIAS_Code number(20), -- код ФИАС (может менятся со временем)
  FIAS_Level number(2), -- уровень распознавания ФИАС
  KLADR_Code number(20), -- код КЛАДР
```

```
  KLADR_Level number(2), -- уровень распознавания КЛАДР
  OKATO number(20), -- код ОКАТО
  OKTMO number(20), -- код ОКТМО
  house_match VARCHAR2(50 CHAR), -- строка соответствия номеру
   строения
  flat_match VARCHAR2(50 CHAR),-- строка соответствия номеру квартиры
  address_post_index number(20), -- почтовый индекс
  house_variants T_HOUSE_VARIANT --варианты возможных записей о домах
);
```

*norm_line*

Нормализованная строка адреса.

*norm_line_lev_dist*

Отклонение нормализованного адреса от исходного: 0 – адреса совпадают, 100 – максимально удалены друг от друга (различаются).

*FIAS_GUID*

GUID в базе ФИАС.

*FIAS_Code*

Код ФИАС.

*FIAS_Level*

Уровень распознавания ФИАС.

*KLADR_Code*

Код в базе КЛАДР.

*KLADR_Level*

Уровень распознавания КЛАДР.

*OKATO*

Код ОКАТО.

*OKTMO*

Код ОКТМО.

*house_match*

Строка соответствия номеру строения (точно или нет).

*flat_match*

Строка соответствия номеру квартиры (точно или нет).

*address_post_index*

Почтовый индекс.

*house_variants*

Массив вариантов возможных записей о домах. Подробнее см. Массив вариантов записей о домах.

### Тип таблицы вариантов поиска в ФИАС T_FIAS_VARIANT

Объекты данного типа предназначены для хранения информации о всех найденных вариантах по данному адресному запросу.

```
CREATE OR REPLACE TYPE "T_FIAS_VARIANT" is table of R_FIAS_VARIANT;
```

# Функции предварительной обработки адресов

## Пакет PKG_PREPROC_AUX

Используется для предварительной фильтрации входных данных для устранения регулярных ошибок источников адресной информации. Позволяет задавать цепочки правил предварительной обработки элементов структуры входных адресов, правила хранятся в таблице PREPROC_RULES.

Виды правил фильтрации (таблица PREPROC_RULETYPES):

- Регулярные выражения;
- Трансляция символов.

Виды элементов адреса (таблица PREPROC_FIELDTYPES):

- Предварительные преобразования, общие для всех полей;
- Заключительные преобразования, общие для всех полей;
- Полная строка адреса;
- Страна;
- Субъект;
- Регион;
- Город;
- Район города;
- Улица;
- Дом;
- Квартира.

Правила фильтрации позволяют скорректировать адресные элементы перед или после отправки в модуль разбора.

Пакет позволяет организовать цепочки применения правил фильтрации. Для цепочек доступны функции редактирования: можно добавлять, изменять, удалять правила, а также менять порядок применения правил в цепочке.

Пакет представляет собой самостоятельный инструмент для обработки строковых данных.

```
create or replace package PKG_PREPROC_AUX is

   -- константы идентификаторов таблицы типов правил
   (PREPROC_RULETYPES.RTYPE_ID)
  g_rtype_REPLACE CONSTANT number(1) := 1; -- правило применения
   функции REGEXP_LIKE
  g_rtype_TRANSLATE CONSTANT number(1) := 2; -- правило применения
   функции TRANSLATE

  -- константы идентификаторов таблицы типов полей преобразований
   элементов адреса (PREPROC_FIELDTYPES.FTYPE_ID)
  g_ftype_ALLPRE CONSTANT number(2) := 1; -- предварительные
   преобразования, общие для всех полей
  g_ftype_ALLPOST CONSTANT number(2) := 2; -- заключительные
   преобразования, общие для всех полей
  -- остальные соответствуют одноименных полям структуры
   pkg_fx_addr.r_RCO_RFM_IN
  g_ftype_ADDRESS CONSTANT number(2) := 3; -- полная строка адреса
  g_ftype_COUNTRY CONSTANT number(2) := 4; -- страна
  g_ftype_SUBJECT CONSTANT number(2) := 5; -- субъект
  g_ftype_REGION CONSTANT number(2) := 6; -- регион
  g_ftype_CITY CONSTANT number(2) := 7; -- город
  g_ftype_DISTRICT CONSTANT number(2) := 8; -- район города
  g_ftype_STREET CONSTANT number(2) := 9; -- улица
  g_ftype_HOUSE CONSTANT number(2) := 10; -- дом
  g_ftype_FLAT CONSTANT number(2) := 11; -- квартира

  --
  -- процедуры манипулирования записями таблицы правил преобразования
   элементов адреса (PREPROC_RULES)
  -- !! не производят COMMIT
  --

  -- добавить новое правило в конец цепочки преобразований для типа
   поля
  function rule_add(
    p_ftype_id number, -- тип поля g_ftype_*
    p_rtype_id number, -- тип правила g_rtype_*
    p_arg1 varchar2, -- аргумент №1
    p_arg2 varchar2 -- аргумент №2
  )
   return PREPROC_RULES.ORDERNUM%type; -- порядковый номер правила в
   цепочке преобразований

   -- модифицировать существующее правило
  procedure rule_upd(
   p_ftype_id number, -- PK: тип поля g_ftype_*
    p_ordernum number, -- PK: порядковый номер правила в цепочке
   преобразований
    p_rtype_id number, -- новый тип правила g_rtype_*
    p_arg1 varchar2, -- новый аргумент №1
    p_arg2 varchar2 -- новый аргумент №2
  );

   -- удалить существующее правило
  procedure rule_del(
    p_ftype_id number, -- PK: тип поля g_ftype_*
    p_ordernum number  -- PK: порядковый номер правила в цепочке
   преобразований
  );
```

```
  -- изменить порядковый номер правила в цепочке преобразований
  procedure rule_move
  (
    p_ftype_id number, -- PK: тип поля g_ftype_*
    p_ordernum_from number, -- PK: порядковый номер правила в цепочке
   преобразований
    p_ordernum_to number -- новый порядковый номер правила.
      -- если указан NULL, то правило перемещается в самый конец
   цепочки
  );

end PKG_PREPROC_AUX;
```

В пакете описаны типы правил фильтрации, типы (адресных полей) фильтрации и процедуры и функции, реализующие работу с цепочками правил фильтрации. Для каждого типа `ftype` можно задать цепочку правил фильтрации. Каждое правило фильтрации в цепочке получает свой порядковый номер ordernum. Применяются цепочки правил с помощью процедуры **`PKG_PREPROC.apply_field_full`**, см. Пакет PKG PREPROC.

*g_rtype_REPLACE*

Тип правила фильтрации с применением функции Oracle **`REGEXP_LIKE`**.

*g_rtype_TRANSLATE*

Тип правила фильтрации с применением функции Oracle **`TRANSLATE`**.

*g_ftype_ALLPRE*

Тип адресного поля для фильтрации: предварительная фильтрация для всех полей. Цепочка преобразований, заданная этим типом, будет выполнена перед остальными правилами фильтрации. (При вызове **`PKG_PREPROC.apply_field_full`**, см. Пакет PKG PREPROC.)

*g_ftype_ALLPOST*

Тип адресного поля для фильтрации: фильтрация после для всех полей. Цепочка преобразований, заданная этим типом, будет выполнена после остальных правил фильтрации. (При вызове **`PKG_PREPROC.apply_field_full`**, см. Пакет PKG PREPROC.)

*g_ftype_ADDRESS*

Тип адресного поля для фильтрации: полная строка адреса. Цепочка преобразований, заданная этим типом, будет выполнена для полной строки адреса. (При вызове **`PKG_PREPROC.apply_field_full`**, см. Пакет_PKG_PREPROC.)

*g_ftype_COUNTRY*

Тип адресного поля для фильтрации: страна. Цепочка преобразований, заданная этим типом, будет выполнена для поля страна. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет_PKG_PREPROC.)

*g_ftype_SUBJECT*

Тип адресного поля для фильтрации: субъект. Цепочка преобразований, заданная этим типом, будет выполнена для поля субъект. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_REGION*

Тип адресного поля для фильтрации: регион. Цепочка преобразований, заданная этим типом, будет выполнена для поля регион. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_CITY*

Тип адресного поля для фильтрации: город. Цепочка преобразований, заданная этим типом, будет выполнена для поля город. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_DISTRICT*

Тип адресного поля для фильтрации: район. Цепочка преобразований, заданная этим типом, будет выполнена для поля район. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_STREET*

Тип адресного поля для фильтрации: улица. Цепочка преобразований, заданная этим типом, будет выполнена для поля улица. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_HOUSE*

Тип адресного поля для фильтрации: дом. Цепочка преобразований, заданные этим типом, будут выполнена для поля дом. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

*g_ftype_FLAT*

Тип адресного поля для фильтрации: квартира. Цепочка преобразований, заданная этим типом, будут выполнена для поля квартира. (При вызове **PKG_PREPROC.apply_field_full**, см. Пакет PKG PREPROC.)

**rule_add**

Функция добавляет правило в цепочку преобразований.

```
function rule_add(
```

```
    p_ftype_id number, -- тип поля g_ftype_*
    p_rtype_id number, -- тип правила g_rtype_*
    p_arg1 varchar2, -- аргумент №1
    p_arg2 varchar2 -- аргумент №2
  )
   return PREPROC_RULES.ORDERNUM%type; -- порядковый номер правила в
   цепочке преобразований
```

**Параметры**
*p_ftype_id*

[вх] Тип адресного поля для фильтрации.

*p_rtype_id*

[вх] Тип правила фильтрации.

*p_arg1*

[вх] Аргумент №1 для применяемой функции заданного типа правила фильтрации. (Например, регулярное выражение для `g_rtype_REPLACE`.)

*p_arg2*

[вх] Аргумент №2 для применяемой функции заданного типа правила фильтрации. (Например, строка замены для `g_rtype_REPLACE`.)

**Возвращаемое значение**
Функция возвращает порядковый номер добавленного правила в цепочке преобразований.

**Пример: добавление правил фильтрации**

```
-- пример добавления правил предварительной фильтрации
-- для компонентов адреса
--
declare

l_ordernum PREPROC_RULES.ORDERNUM%type;

begin

-- удалить все нецифровые символы перед номером дома:
l_ordernum := PKG_PREPROC_AUX.rule_add
(
                p_ftype_id => PKG_PREPROC_AUX.g_ftype_HOUSE,
                p_rtype_id => PKG_PREPROC_AUX.g_rtype_REPLACE,
                p_arg1 => '^[^[:digit:]]+',
                p_arg2 => ''
);

-- удалить все нецифровые символы перед номером квартиры:
l_ordernum := PKG_PREPROC_AUX.rule_add
(
                p_ftype_id => PKG_PREPROC_AUX.g_ftype_FLAT,
                p_rtype_id => PKG_PREPROC_AUX.g_rtype_REPLACE,
                p_arg1 => '^[^[:digit:]]+',
                p_arg2 => ''
);
```

```
end;
/
commit;
```

**rule_upd**

Процедура модифицирует существующее правило в цепочке преобразований.

```
procedure rule_upd(
   p_ftype_id number, -- PK: тип поля g_ftype_*
   p_ordernum number, -- PK: порядковый номер правила в цепочке
   преобразований
   p_rtype_id number, -- новый тип правила g_rtype_*
   p_arg1 varchar2, -- новый аргумент №1
   p_arg2 varchar2 -- новый аргумент №2
);
```

**Параметры**
*p_ftype_id*

[вх] Тип адресного поля для фильтрации.

*p_ordernum*

[вх] Порядковый номер правила в цепочке преобразований (полученный при выполнении функции **rule_add** или установленный процедурой **rule_move**).

*p_rtype_id*

[вх] Тип правила фильтрации.

*p_arg1*

[вх] Аргумент №1 для применяемой функции заданного типа правила фильтрации. (Например, регулярное выражение для `g_rtype_REPLACE`.)

*p_arg2*

[вх] Аргумент №2 для применяемой функции заданного типа правила фильтрации. (Например, строка замены для `g_rtype_REPLACE`.)

**rule_del**

Процедура удаляет существующее правило из цепочки преобразований.

```
  procedure rule_del(
    p_ftype_id number, -- PK: тип поля g_ftype_*
    p_ordernum number  -- PK: порядковый номер правила в цепочке
   преобразований
  );
```

**Параметры**
*p_ftype_id*

[вх] Тип адресного поля для фильтрации. Определяет из какой цепочки правил фильтрации будет удаляться правило.

*p_ordernum*

[вх] Порядковый номер удаляемого правила в цепочке преобразований (полученный при выполнении функции **rule_add** или установленный процедурой **rule_move**).

**rule_move**

Процедура изменяет порядковый номер существующего правила в цепочке преобразований.

```
procedure rule_move(
   p_ftype_id number, -- PK: тип поля g_ftype_*
   p_ordernum_from number, -- PK: порядковый номер правила в цепочке
   преобразований
   p_ordernum_to number -- новый порядковый номер правила.
);
```

**Параметры**
*p_ftype_id*

[вх] Тип адресного поля для фильтрации. Определяет, в какой цепочке правил будет изменяться порядок применения правил.

*p_ordernum_from*

[вх] Порядковый номер правила, которое необходимо переместить, в цепочке преобразований (полученный при выполнении функции **rule_add** или установленный процедурой **rule_move**).

*p_ordernum_to*

[вх] Новый порядковый номер в цепочке преобразований. Если указан NULL, то правило перемещается в самый конец цепочки

## Пакет PKG_PREPROC

Пакет применяет правила фильтрации к соответствующему элементу адреса.

```
create or replace package PKG_PREPROC is

/*
--
-- применить все фильтры к соответствующим полям структуры адреса
--

   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_ADDRESS, p_value
   => p_fields.address_line);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_COUNTRY, p_value
   => p_fields.Part_Country);
```

```
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_SUBJECT, p_value
   => p_fields.Part_Subject);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_REGION, p_value
   => p_fields.Part_Region);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_CITY, p_value =>
   p_fields.Part_City);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_DISTRICT,
   p_value => p_fields.Part_District);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_STREET, p_value
   => p_fields.Part_Street);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_HOUSE, p_value
   => p_fields.house_line);
   apply_field_full(p_idx => PKG_PREPROC_AUX.g_ftype_FLAT, p_value =>
   p_fields.flat_line);
*/
   procedure apply_field_full(
   p_idx pls_integer,
   p_value in out varchar2
);

  -- перечитать кэш правил из таблицы PREPROC_RULES
  -- выполняется автоматически при инициализации данного пакета
  procedure init;

end PKG_PREPROC;
```

В пакете объявлена процедура, выполняющая фильтрацию, **apply_field_full** и в комментариях приведены примеры ее использования.

### apply_field_full

Процедура применяет заданную цепочку правил к переданной переменной, модифицируя ее последовательно, согласно правилам цепочки. Кроме того, перед заданной параметром `p_idx` цепочкой всегда применяется цепочка правил предобработки `g_ftype_ALLPRE`, а после этих 2х цепочек применяется цепочка правил постобработки `g_ftype_ALLPOST`.

```
   procedure apply_field_full(
   p_idx pls_integer, -- индекс типа фильтрации
   p_value in out varchar2 -- значение фильтра
);
```

**Параметры**

*p_idx*

[вх] Индекс типа фильтрации. Необходимо задать одно из объявленных в константах `g_ftype_...` в пакете **PKG_PREPROC_AUX** значений. Определяет какую цепочку правил следует применить к переданному значению.

*p_value*

[вх вых] параметр, к которому необходимо применить цепочку правил.

# Вспомогательные пакеты и таблицы

Не предназначены для конечного пользователя API, их структура может со временем меняться.

### Настройки системы

В таблице TBLSETTINGS, пакета PKG_SETTINGS хранятся настройки системы, в частности, URL сервера SOAP сервиса по умолчанию (параметр “URL_DEFAULT”).

### Логи системы

Таблица TBLTRACERN содержит логи работы Oracle API для мониторинга возможных проблем администратором.

В схеме существует задание ежедневного выполнения “DAILY”, которое каждую ночь вызывает процедуру PRDN_DAILY для удаления записей лога старше 14 дней.

Успешность выполнения этого задания также логируется.

### Реализация протокола SOAP для обращения к Службе разбора адресов

За взаимодействие по протоколу SOAP с Адаптером Службы разбора адресов через HTTP отвечают пакеты PKG_FX, RCO_HELPER.

### Работа с форматом данных JSON

За работу с данными в формате JSON отвечает свободно распространяемая библиотека PL\SQL “PLJSON-2.5.1”.

# Пример обработки адреса

В примере создается таблица для кэшированных адресов, разбор которых произведен в течении последних суток, и функция, принимающая в качестве входного параметра адрес, и возвращающая первый (наиболее близкий) нормализованный вариант. Если близкий вариант не найден (отклонение больше `l_max_distance`), возвращается адрес, составленный из нормализаций адресных элементов (нормализация по частям).

```
--Таблица для кэширования нормализованных строк
CREATE TABLE ADDR_CACHE(
   "STR_IN" VARCHAR2(4000), -- входная адресная строка
   "STR_ADDR" VARCHAR2(4000), -- нормализованный результат
   "DT" DATE DEFAULT sysdate  -- дата-время записи
   )
/

create or replace function rco_parse_address (
   s varchar2 --Входная строка для нормализации
) return varchar2
as
  l_id int;
```

```
  l_u_s varchar2(4000) := upper(s);
  l_addr_out varchar2(4000);
  l_addr_in RCO_AP3.R_RCO_ADDR_IN := RCO_AP3.R_RCO_ADDR_IN();
  l_out_multi RCO_AP3.T_RCO_ADDR_OUT;
  l_out RCO_AP3.R_RCO_ADDR_OUT;
  l_variant RCO_AP3.R_FIAS_VARIANT;
  l_have_variants int := 0;
  l_distance int := 0;
  l_max_distance constant int := 30; -- максимальное отклонение
   нормализованной строки
pragma autonomous_transaction;
begin
  begin
    select str_addr into l_addr_out from ADDR_CACHE where str_in =
   l_u_s and dt > sysdate -1;
    exception when no_data_found then
      l_addr_in.addrAddFull(p_try_fuzzy => true,p_address_line => s);
      l_out_multi := RCO_AP3.pkg_fx_addr.fx_get_addr(l_addr_in);
      l_out := l_out_multi(1);
      if l_out.error is null then
           if not l_out.variants is null then
                if l_out.variants.count > 0 then
                     l_variant := l_out.variants(1);
                     l_distance := l_variant.norm_line_lev_dist;
                     if l_distance < l_max_distance then
                          l_have_variants := 1;
                     end if;
                end if;
           end if;
           if l_have_variants = 1 then
                l_addr_out := l_variant.norm_line;
           else
                --нормализация по частям
                l_addr_out := l_out.norm_parts_line;
           end if;
           delete ADDR_CACHE where str_in = l_u_s;
           insert into ADDR_CACHE (str_in, str_addr)
                values (l_u_s, l_addr_out);
           commit;
      else
           l_addr_out := l_u_s;
           DBMS_OUTPUT.put_line ('Адрес не нормализован '||
                 s ||' "'||l_out.error||'"'
           );
      end if;
  end;
return l_addr_out;
end;
/
```