ОПИСАНИЕ API платформы АрхиГраф

ООО «ТриниДата», 2015-2022

Содержание

1. Назначение платформы АрхиГраф

Платформа АрхиГраф предназначена для хранения информационной модели, нормативно-справочной информации, основных и транзакционных данных организации. Она обеспечивает доступ прикладным программным компонентам программного как к данным, размещенных в хранилищах под ее управлением, так и к любой информации во внешних хранилищах.

Платформа АрхиГраф может выполнять функции классической MDM-системы на предприятии, или использоваться в качестве ядра сложных прикладных автоматизированных систем, в том числе интегрированных аналитических систем и логических витрин данных, систем построения отчетности и др. Также АрхиГраф может выступать в роли корпоративного инструмента управления данными (Data Governance) в сложных, интегрированных инфраструктурах, включающих десятки бизнес-приложений.

АрхиГраф обеспечивает синхронизацию данных между различными приложениями, контроль прав доступа при обращении к информации, версионность модели и данных, извлечение данных из внешних источников в режиме логической витрины, подписку на уведомления об изменении модели и данных, форматно-логический контроль информации и многие другие функции.

Структура (модель) информации, с которой работает АрхиГраф, представляется в виде онтологической модели. Это позволяет приложениям-клиентам работать с машинно-читаемым представлением не только самих данных, но и их структуры – информационной модели. Онтологии допускают применение множественной (фасетной) классификации сущностей, наследование наборов атрибутов, множественность значений атрибутов и др.

2. Архитектура обмена

2.1. Форматы запросов

Принцип работы программного интерфейса АрхиГраф основан на обмене пакетами с другими программными системами. Поддерживаются запросы в двух основных форматах:

- JSON
- XML

Запросы в таких форматах могут выполняться в синхронном режиме, как HTTP-запросы, или в асинхронном – с помощью очередей RabbitMQ или Kafka.

Также поддерживается REST API, через который доступны все функции основного API (см. раздел 6 настоящего документа).

2.2. Транспорт

Транспорт XML или JSON пакетов основного API может осуществляться несколькими способами:

- при помощи очередей RabbitMQ или Kafka (предпочтительный способ, и единственный возможный в высоконагруженных и высоконадежных архитектурах),
- HTTP-сервисов,
- web-интерфейса (в отладочных целях).

Форматы пакетов основного API и последовательность обмена ими не зависят от способа передачи информации. В процессе обмена каждое приложение, желающее отправить запрос к АрхиГраф, должно сформировать сообщение в любом из поддерживаемых форматов, и отправить его в систему при помощи соответствующего транспорта. Платформа возвратит ответ при помощи того же транспортного протокола. Исходящий пакет будет возвращен в том же формате, в каком получен входящий.

3. Средства управления платформой АрхиГраф

Настройка модели (структуры) данных для работы АрхиГраф.MDM выполняется с помощью редактора онтологий АрхиГраф.Мир.

Создание правил форматно-логического контроля, правил дополнения информации и поиска дубликатов, работа с качеством данных, поиск информации пользователем, работа с бизнес-глоссарием происходят в среде АрхиГраф.СУЗ.

Настройка точек доступа и хранилищ АрхиГраф.MDM, управление правами доступа к данным и модели, создание шаблонов маршрутов согласований выполняются в Интерфейсе администрирования АрхиГраф.MDM.

Общие вопросы использования АрхиГраф.MDM рассматриваются в Руководстве пользователя АрхиГраф.MDM.

Для отладки и настройки работы с платформой АрхиГраф полезна также форма тестирования XML- и JSON HTTP-запросов основного API, доступная по адресу /mdm в любой инсталляции MDM. Форма имеет такой вид:

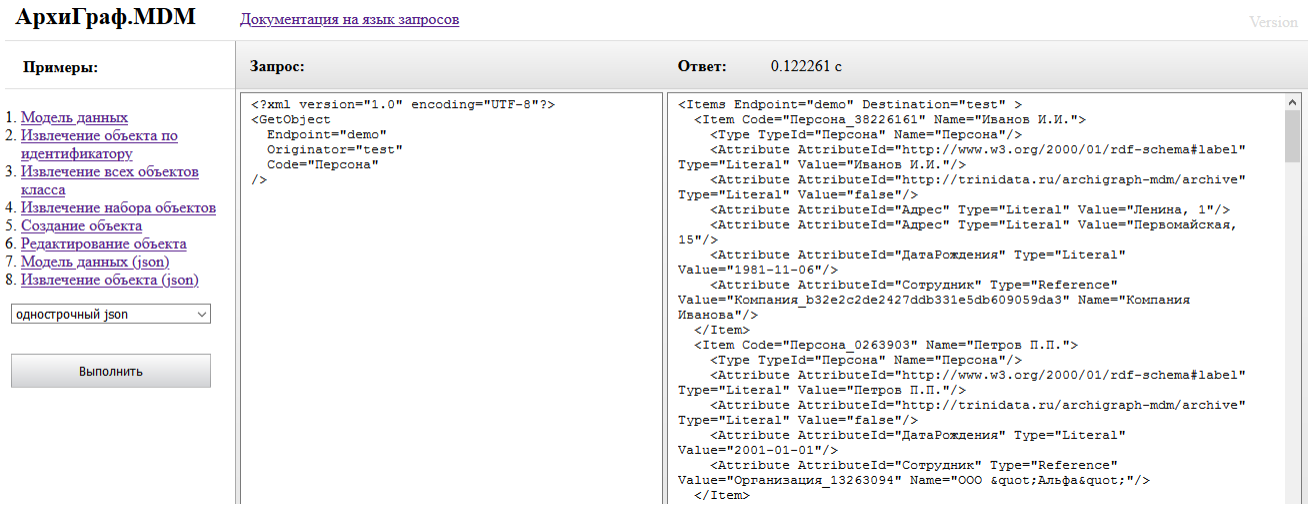

Рис. 2. Форма тестирования HTTP-запросов к платформе АрхиГраф

Справа расположены ссылки на примеры запросов. Нажатие на любую ссылку копирует текст запроса в поле «Запрос», где его можно изменить. Нажатие на кнопку «Отправить» выполняет обращение к платформе, ответ отображается в поле «Ответ».

Для выполнения HTTP-запросов к системе со стороны программных компонентов нужно отправить POST-запрос на URL /mdm, передав в параметре request тело XML- или JSON-запроса.

4. Основной API запросов и ответов АрхиГраф

4.1. Общие сведения

Пакеты в формате XML и JSON имеют схожую структуру.

Пример пакета в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetDataSchema Endpoint="demo" Originator="test" />
```

Тот же пакет в формате JSON:

```
{"GetDataSchema": {"Endpoint":"demo", "Originator":"test"}}
```

Названия тегов и атрибутов не зависят от регистра.

Любые запросы, независимо от корневого тега, могут иметь следующие стандартные параметры:

Endpoint – код точки доступа[1], к которой относится запрос. Применим ко всем запросам за исключением запроса на получение списка точек доступа (GetEndpoints). Параметр не обязателен: если код точки доступа не передан, запрос будет отнесен к точке, определенной настройками системы как точка доступа по умолчанию. Если значение параметра передано во входном пакете, то это значение будет возвращено в корневом теге ответа в атрибуте Endpoint.

OperationId – идентификатор запроса. Применим ко всем запросам. Если OperationId передан во входном пакете, то его значение будет возвращено в атрибуте OperationId корневого тега ответа. Не обязательный, но необходим при обмене пакетами посредством очередей, так как позволяет сопоставить исходящий запрос с относящимся к нему ответом.

Originator – код запрашивающей информационной системы-клиента. Применим ко всем запросам. Если параметр передан в запросе, то его значение будет возвращено в атрибуте Destination корневого тега ответа. Информационная система с переданным кодом должна быть зарегистрирована в системе, и для нее должны быть настроены права доступа. В общем случае Originator не обязателен, но при запросе без указания системы будут применяться самые минимальные права доступа (права доступа для анонимных систем).

[1] Точка доступа (endpoint) представляет собой обособленную часть логического пространства данных. Можно уподобить точки доступа отдельным базам данных в реляционной СУБД.

Token – применим ко всем запросам. Строка служит для обеспечения информационной безопасности: для идентификации системы проверяется соответствие кода системы, переданного в атрибуте Originator, с зарегистрированным для него токеном. Информационная система может быть зарегистрирована с пустым токеном, в этом случае параметр Token передавать не обязательно.

User – применим ко всем запросам. Необязательный строковый параметр. Рекомендуется в качестве значения параметра User передавать имя пользователя, действие которого в приложении вызвало отправку данного запроса. Переданное значение имеет справочный характер, будет зафиксировано в логах MDM, но не повлияет на выполнение запроса.

Comment – применим ко всем запросам. Необязательный параметр: произвольный текстовый комментарий к запросу. Переданное значение будет запомнено в логах MDM, не влияет на выполнение запроса.

ReturnTime – флаг, принимающий значения 0 или 1, применим ко всем запросам. Если передано значение ReturnTime=1, то в параметре ExecuteTime корневого тега ответа будет возвращено время выполнения запроса в секундах.

ReturnTimeLog - флаг, принимающий значения 0 или 1. В случае передачи значения 1, ответ будет дополнен логом времени выполнения, содержащимся в теге TimeLogs. Каждая строка лога содержит информацию о модуле (Module), методе модуля (Method), числе вызовов данного метода (Count), общем (TotalTime) и среднем (AverageTime) времени работы метода в секундах, а так же о количестве использований кеша (CacheUsed).
Пример ответа, содержащего лог времени выполнения:

```
<Items Count="1" Destination="test" CacheUsed="1" >
  <Item Code="ИвановИИ" Name="Иванов И.И.">
    <Type TypeId="Персона" Name="Персона"/>
    ...
  </Item>
  <TimeLogs>
    <TimeLog Module="mdm_parser" Method="Handle" Count="1"
TotalTime="0.19787216186523" AverageTime="0.09787216186523"/>
    <TimeLog Module="mdm_parser" Method="Handle_prepare" Count="1"
TotalTime="0.0088210105895996" AverageTime="0.0018210105895996"/>
    . . .
    <TimeLog Module="mdm_storage_fuseki" Method="tdb_query" Count="2"
TotalTime="0.048786163330078" AverageTime="0.024393081665039"/>
    <TimeLog Module="mdm_storage_fuseki" Method="getName" Count="1"
TotalTime="1.1205673217773E-5" AverageTime="1.1205673217773E-5"
CacheUsed="1"/>
  </TimeLogs>
</Items>
```

Флаги **ClearCache** и **UseCache** позволяют на уровне выполнения запроса управлять использованием кеша. При передаче ClearCache=1 по всем найденным объектам произойдет сброс кеша (если хранилище объекта использует его). UseCache=0 позволяет отключить использование кеша на время выполнения конкретного запроса.

Атрибуты ReturnTimeLog, ClearCache и UseCache используются главным образом на этапе отладки для уточнения вероятных причин медленной или некорректной работы запросов.

В случае ошибки при обработке любого запроса в ответ платформа возвращает пакет InvalidPackage.

Пакет **InvalidPackage** – сообщение о неверном запросе. Атрибуты пакета InvalidPackage:

Message - текст сообщения об ошибке

ErrorCode – целочисленный код ошибки

Пример пакета в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<InvalidPackage Message="Object not found" ErrorCode="202"
Destination="test" />
```

Пример пакета в формате JSON:

```
{"InvalidPackage":{ "Message":"Object not found", "ErrorCode":"202",
"Destination":"test" }}
```

4.2. Запросы для работы с точками доступа и хранилищами

4.2.1. Получение набора точек доступа

GetEndpoints – запрос на получение списка точек, доступ к которым разрешен для заданной информационной системы-клиента платформы.

Параметры

Не имеет собственных дополнительных параметров.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetEndpoints Originator="test" />
```

В формате JSON:

```
{"GetEndpoints":{"Originator":"test"}}
```

Ответ

Пакет **Endpoints**

Вложенные теги: *Endpoint*

Тег Endpoint содержит описание отдельной точки доступа.

Атрибуты тега Endpoint:

Code – код точки доступа. Используется в качестве значения атрибута Endpoint входящих пакетов;
Name - читаемое наименование точки доступа;
Default – является ли точкой доступа по умолчанию, значение true либо false.

Пример

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Endpoints Destination="test">
  <Endpoint Code="demo" Name="Демонстрационная точка доступа" Default="true" />
</Endpoints>
```

В формате JSON:

```
{
   "Endpoints":
    {
    "Destination": "test",
    "Endpoint": [
        {
            "Code":"demo",
            "Name":"Демонстрационная точка доступа",
            "Default":"true"
        }
     ]
  }
}
```

4.2.2. Получение списка хранилищ

GetStorages – запрос на получение списка хранилищ с их характеристиками: набор попадающих классов, хранение истории, поддержка 4D-историйности, работа в режиме логической витрины данных (LDW), тип физического хранилища.

Параметры

В случае, если в запросе передан атрибут **EndpointCode**, будет возвращен список хранилищ, относящихся только к указанной точке доступа. Если код точки доступа не указан, будут возвращены хранилища всех точек, доступных информационной системе.

Для каждого хранилища возвращаются его параметры (код, наименование, реквизиты доступа и т. п.), а также список классов модели, объекты которого содержатся в указанном хранилище. По умолчанию будут возвращены классы верхнего уровня.

Атрибут **WithSubclasses,** принимающий значения 0 и 1, позволяет получить полный перечень классов, с учетом наследуемых. По умолчанию WithSubclasses имеет значение, равное 0.

Если получать список классов не требуется, запрос можно выполнить с передачей флага **WithoutClasses**=1. В этом случае будут возвращены только параметры хранилищ. Без привязки к классам модели.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetStorages EndpointCode="demo" Originator="test" WithSubclasses="1" />
```

В формате JSON:

```
{
  "GetStorages":{
     "EndpointCode":"demo",
     "Originator":"test",
     "WithSubclasses": "1"
  }
}
```

Ответ

Пакет **Storages**

Описание исходящего пакета Storages

Вложенные теги:

Storage – содержит описание хранилища данных

ObjectType – характеризует класс модели, объекты которого хранятся в данном хранилище

Тег Storage

Атрибуты тега Storage:

Code – код (идентификатор) хранилища в АрхиГраф
Name – читаемое наименование хранилища
Endpoint – код точки доступа, к которой относится хранилище
History – флаг, принимает значения 0 и 1. Показывает, запоминается ли для данного хранилища средствами АрхиГраф история изменения объектов, в нем содержащихся.
Timestamp – флаг, принимает значения 0 и 1. Определяет, поддерживает ли хранилище 4D-историйность данных. Такая поддержка обеспечивается, например, в HBase за счет хранения множества значений каждого атрибута объекта, причем каждое значение помечено timestamp.
Logic – флаг, принимает значения 0 и 1. Поддерживает ли хранилище работу с правилами логического вывода.
Type – тип физического хранилища (PostgreSQL, MongoDB и т.п.)
Model – флаг, принимает значения 0 и 1. Является ли хранилище хранилищем модели. Хранилище модели содержит описание классов и атрибутов модели, а также является хранилищем по умолчанию для индивидуальных объектов: если для класса в не указано хранилище явно, то объекты, принадлежащие данному классу, будут ассоциированы с хранилищем модели.
Editable – флаг, принимает значения 0 и 1. Является ли хранилище редактируемым, т.е. возможно ли создание/изменение/удаление объектов в данном хранилище с помощью запросов MDM, или же хранилище для MDM предоставляет доступ только на чтение.
Multilang – флаг, значения 0 и 1. Поддерживает ли хранилище мультиязычные данные.

Place – строковый атрибут - наименование площадки, на которой размещено хранилище. Не обязателен, если значение не задано, то атрибут не возвращается. В случае, если для некоторых хранилищ задано размещение, применение запросов на извлечение и редактирование объектов можно сузить только до заданной площадки.

Атрибуты тега ObjectType:

Code – код АрхиГраф класса модели
Name – читаемое наименование класса

Пример ответа

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Storages Destination="test">
  <Storage Code="storage1" Name="TDB для хранения модели" Endpoint="demo"
History="1" Timestamp="0" Logic="1" Type="Fuseki" Editable="1" Multilang="1"
/>
  <Storage Code="storage2" Name="БД контрагентов" Endpoint="demo"
History="0" Timestamp="0" Logic="1" Type="MongoDB" Editable="1"
Multilang="1">
     <ObjectType Code="Контрагент" Name="Контрагенты" />
  </Storage>
</Storages>
```

В формате JSON:

```
{"Storages:
  {
    "Destination":"test",
    "Storage": [
      {
        "Code":"storage1",
        "Name": "TDB для хранения модели",
        "Endpoint": "demo",
        "History":"1",
        "Timestamp":"0",
        "Logic":"1",
        "Type": "Fuseki",
        "Editable":"1",
        "Multilang":"1"
      },
      {
        "Code":"storage2",
        "Name": "БД контрагентов",
        "Endpoint": "demo",
        "History":"0",
        "Timestamp":"0",
        "Logic":"1",
        "Type":" MongoDB",
        "Editable":"1",
        "Multilang":"0"
        "ObjectType": [
          {"Code":"Контрагент", "Name":"Контрагенты"}
        ]
      }
    ]
  }
}
```

4.2.3. Изменение параметров хранилища

UpdateStorage – запрос на редактирование изменяемых параметров хранилища.

Параметры

Вложенные теги: *Storage*.

Атрибуты тега Storage:

Code – обязателен, код изменяемого хранилища в АрхиГраф
Name – не обязателен, строка. Наименование хранилища.
History – не обязателен, 0 или 1. Отменить или установить ведение истории изменения объектов средствами АрхиГраф.
AddTypes – не обязателен, 0 или 1. Если передано значение 0 или атрибут не задан, то список классов, относящихся к хранилищу, будет заменен на список классов, переданных в текущем пакете в теге ObjectType. Если же передано значение 1, то список классов, относящихся к хранилищу, будет дополнен переданными классами.
DeleteTypes – не обязателен, 0 или 1. Если передано значение 1, то из списка классов, относящихся к хранилищу, будут исключены классы, переданные в пакете.

Вложенные теги: *ObjectType*

Атрибуты тега ObjectType:

Code – код АрхиГраф класса модели

Пример запроса

В формате XML:

```
<UpdateStorage Endpoint="demo" Originator="test">
  <Storage Code="storage2" AddTypes="1">
  <ObjectType Code="Подрядчик" />
  </Storage>
</UpdateStorage>
```

Ответ

OperationResults – пакет с информацией о результате выполнения операции. Подробнее описание пакета см. в разделе описания запроса UpdateObject.

4.2.4. Получение описания хранилища для класса

GetStoragesMapping – получить параметры физического хранения объектов класса (тип и реквизиты БД, коллекция/таблица, имена полей/столбцов соответствующих атрибутам). Может использоваться приложением для работы напрямую с хранилищем данных, например – для создания и выполнения заданий MapReduce и др.

Параметры

Вложенный тег *ObjectType* – классы модели, по которым требуется получить информацию

Атрибуты тега ObjectType:

Code – обязательный, код АрхиГраф класса модели

Пример запроса

В формате XML:

```
<GetStoragesMapping Endpoint="demo" Originator="test">
  <ObjectType Code="Контрагент" />
</GetStoragesMapping>
```

В формате JSON:

```
{"GetStoragesMapping": {
    "Endpoint":"demo",
    "Originator":"test",
    "ObjectType": [{"Code":"Контрагент"}]
  }
}
```

Ответ

Пакет **MapStorages**

Вложенные теги:

MapStorage – описание хранилища, содержащего объекты заданного класса,

MapAttribute – описание физического хранения значений атрибутов,

Target – только для атрибутов ссылочного типа – список классов модели, характеризующий область значения атрибута.

Атрибуты тега MapStorage:

ObjectType – класс модели, переданный в запросе
Code – код (идентификатор) хранилища в АрхиГраф
Name – читаемое наименование хранилища
Type – тип физического хранилища (например, Fuseki, PostgreSQL и др.)
Host, **Port**, **Database** – реквизиты подключения к БД
Table – коллекция/таблица, в которой хранятся объекты

Атрибуты тега MapAttribute (вложен в MapStorage):

AttributeId – идентификатор атрибута в модели
Field – поле/столбец для хранения значения атрибута в БД
Type – тип атрибута. Принимает значения Literal (литерал) либо Reference (ссылка на другой объект)
Datatype – тип для литеральных атрибутов. Принимаемые значения:

- xsd:string – строка
- xsd:boolean – флаг да/нет (true/false)
- xsd:integer – целое число
- xsd:double - число
- xsd:date - дата
- xsd:dateTime - дата и время

Получить перечень всех литеральных типов, поддерживаемых платформой, можно запросив объекты специального класса archigraph:type.

Запрос для получения перечня литеральных типов в формате XML:

```
<GetObjectsGroup Code="archigraph:type" />
```

Запрос для получения перечня литеральных типов в формате json:

```
{"GetObjectsGroup":{"Code":"archigraph:type"}}
```

Атрибуты тега Target (вложен в MapStorage):

Code – код класса

Name – читаемое наименование класса

Пример ответа

В формате XML:

```
 <MapStorages Endpoint="demo" Destination="test">
  <MapStorage ObjectType="Контрагент" Code="storage2" "Name"="БД
контрагентов" Type="MongoDB" Host="12.12.12.123" Port="27017"
Database="demo" Table="clients">
    <MapAttribute AttributeId="ИНН" Field="inn" Type="Literal"
Datatype="xsd:string" />
    <MapAttribute AttributeId="Город" Field="city" Type="Reference">
    <Target TargetId="СправочникГород" Name="Города" />
  </MapAttribute>
  </MapStorage>
  <MapStorage ObjectType="СправочникГород" Code="storage1" Type="Fuseki"
Host="12.12.12.123" Port="8080" Table="demo" />
</MapStorages>
```

4.3. Запросы для работы с моделью

4.3.1. Получение информационной модели

GetDataSchema – запрос на получение структуры информационной модели.

Параметры

StartElement - стартовый класс интересующего фрагмента модели. Если параметр пропущен, возвращается содержимое всей модели.

WithoutSubClasses – необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. Имеет смысл только при указании StartElement. Если значение атрибута 0 или не задано, то возвращается описание модели как самого класса, заданного в атрибуте StartElement, так и всех его подклассов. Если же значение WithoutSubClasses=1, то возвращается описание только самого класса, указанного в StartElement.

WithoutInherited - необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. Если значение 0 или не задано, то описание класса содержит все атрибуты, применимые к объектам данного класса, включая атрибуты, унаследованные

от родительских классов. Если же WithoutInherited=1, то для каждого класса будут возвращены только атрибуты, ассоциированные непосредственно с указанным классом, без наследуемых.

WithoutRangeInherited - необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. Если значение 0 или не задано, то описание области значений атрибутов-ссылок, включает полный перечень классов, включая подклассы. Если же WithoutInherited=1, то для области значений атрибута-ссылки будут возвращены только классы, ассоциированные непосредственно с атрибутом, без вложенных.

WithoutAttributes - необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. При задании WithoutAttributes=1 будет возвращено описание классов без перечня атрибутов.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetDataSchema StartElement="Компании" WithoutSubClasses="1" />
```

Пример запроса в формате JSON:

```
{"GetDataSchema": {"StartElement":"Компании", "WithoutSubClasses": "1"}}
```

Ответ

DataSchema – пакет с описанием структуры информационной модели: перечнем всех описанных в ней классов, атрибутов и связей.

Атрибуты пакета DataSchema

StartElement – корневой элемент фрагмента модели, если был задан в запросе

Prefix – префикс названий элементов модели по умолчанию. Во всех остальных пакетах префикс не указывается, если он соответствует этому префиксу. Теоретически, в пределах одной модели данных могут использоваться разные префиксы. В этом случае идентификаторы элементов модели, имеющих другой префикс, будут иметь полный вид - http://.../.../[Имя элемента].

Тег *ObjectType* – передает информацию о каком-либо классе.

Атрибуты тега ObjectType:

Code – код АрхиГраф класса

Name – читаемое наименование класса

Archive – принимает значение true или false. Значением true помечаются устаревшие классы.

Тег *Parent* – передает информацию о том, что класс является потомком другого класса. Каждый класс может являться потомком любого числа других классов.

Атрибуты тега Parent:

ParentId – уникальный код класса-родителя

Тег *Attribute*

Передает информацию об атрибуте, присущем объектам каком-либо класса. Атрибуты наследуются классами рекурсивно, то есть если атрибут «ИНН» объявлен для класса «Юридические лица», он считается объявленным и для объектов вложенного в него класса «Компании».

Атрибуты тега Attribute

AttributeId – уникальный код атрибута

Name – читаемое наименование атрибута

Type – тип значения атрибута: Literal для атрибутов-литералов, Reference для атрибутов, хранящих ссылки на другие объекты.

DataType – используется для атрибутов-литералов, хранит тип значения. Указывается один из стандартных типов данных xsd: integer, string, date, dateTime, boolean, double.

MinCardinality – минимальное число значений этого атрибута для каждого объекта. Если MinCardinality =1, значит, атрибут обязателен.

MaxCardinality – максимальное число значений этого атрибута для каждого объекта. Атрибуты MinCardinality и MaxCardinalityмогут не указываться, тогда атрибут может принимать любое количество значений.

RangeIntersection – необязательный атрибут. Задается для свойств-ссылок в случае, если областью значений является пересечение нескольких классов (RangeIntersection="true"). По умолчанию, если атрибут RangeIntersection не указан, полагается, что область значения - объединение классов, указанных во вложенных тегах Target.

ClassSet – необязательный атрибут, может быть задан для свойств-литералов и свойств-ссылок. Используется, когда область применения свойства – пересечение нескольких классов. В этом случае в значении ClassSet указываются (через запятую) идентификаторы тех классов, к которым одновременно должен принадлежать объект, чтобы к нему было применимо описываемое свойство.

Archive – принимает значение true или false. Значение true соответствует устаревшим атрибутам, описание которых сохранено для обратной совместимости накопленных данных и текущей модели.

Тег *Target*

Указывает классы, объекты которых могут быть значениями для данного атрибута-ссылки.

Атрибуты тега Target:

TargetId – уникальный код класса

Name – читаемое наименование класса

Пример ответа

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSchema StartElement="Компания" Prefix="http://some-model.ru/instance">
 <ObjectType Code="Компания" Name="Компании">
  <Parent ParentId="ЮридическиеЛица"/>
  <Attribute Type="Literal" AttributeId="Телефон" Name="телефон"
DataType="xsd:string" MinCardinality="1" />
 </ObjectType>
 <ObjectType Code="Персона" Name="Персоны">
  <Parent ParentId="Компания"/>
  <Attribute Type="Literal" AttributeId="ДатаРождения" Name="дата рождения"
DataType="xsd:date" MinCardinality="1" MaxCardinality="1"/>
  <Attribute Type="Reference" AttributeId="РаботаетВ" Name="работает в">
    <Target TargetId="Компания" Name="Компании" />
  </Attribute>
 </ObjectType>
</DataSchema>
```

В формате JSON:

```
{"DataSchema": {
 "StartElement":"Компания",
 "Prefix": "http://some-model.ru/instance",
 "ObjectType": [
  {
    "Code": "Компания",
    "Name": "Компании",
    "Parent": [ {"ParentId": "ЮридическиеЛица"} ],
    "Attribute": [
      {
        "Type": "Literal",
        "AttributeId": "Телефон",
        "Name": "телефон",
        "DataType": "xsd:string",
        "MinCardinality":1
      }
    ]
  },
  {
    "Code": "Персона",
    "Name": "Персоны",
    "Parent": [ {"ParentId": "Компания"} ],
    "Attribute": [
      {
        "Type": "Literal",
        "AttributeId": "ДатаРождения",
        "Name": "дата рождения",
        "DataType": "xsd:date",
        "MinCardinality":1,
        "MaxCardinality":1
      },
      {
        "Type": "Reference",
        "AttributeId": "РаботаетВ",
        "Name": "работает в",
        "Target": [ {"TargetId": "Компания", "Name": "Компании"} ]
```

```
        }
      ]
    }
    ]
  }
}
```

4.3.2. Получение информационной модели в компактной форме

GetDataSchemaCompact – запрос на получение структуры информационной модели в компактной форме

Параметры

StartElement – стартовый класс интересующего фрагмента модели. Если параметр пропущен, возвращается содержимое всей модели.

WithoutSubClasses – необязательный атрибут, принимает значения 0 и 1, значение по умолчанию 0. Имеет смысл только при указании StartElement. Если значение атрибута 0 или не задано, то возвращается описание модели как самого класса, заданного в атрибуте StartElement, так и всех его подклассов. Если же значение WithoutSubClasses=1, то возвращается описание только самого класса, указанного в StartElement.

WithoutInherited – необязательный атрибут, принимает значения 0 и 1, значение по умолчанию 0. Если значение 0 или не задано, то описание класса содержит все атрибуты, применимые к объектам данного класса, включая атрибуты, унаследованные от родительских классов. Если же WithoutInherited=1, то для каждого класса будут возвращены только атрибуты, ассоциированные непосредственно с указанным классом, без наследуемых.

WithoutRangeInherited - необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. Если значение 0 или не задано, то описание области значений атрибутов-ссылок, включает полный перечень классов, включая подклассы. Если же WithoutInherited=1, то для области значений атрибута-ссылки будут возвращены только классы, ассоциированные непосредственно с атрибутом, без вложенных.

WithoutAttributes - необязательный атрибут, может принимать значения 0 и 1, значение по умолчанию 0. При задании WithoutAttributes=1 будет возвращено описание классов без перечня атрибутов.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetDataSchemaCompact StartElement="Компании"/>
```

В формате JSON:

```
{"GetDataSchemaCompact": {"StartElement":"Компании"}}
```

Ответ

Пакет **DataSchemaCompact -** структура модели данных в кратком формате

Пакет аналогичен описанному выше пакету DataSchema, за исключением того, что в нем перечень атрибутов не повторяется для каждого класса. Вместо этого набор атрибутов задается отдельно тегами AttributeDefinition, а для каждого класса указываются применимые к нему атрибуты тегами ApplicableAttribute (внутри тега ObjectType).

Тег *AttributeDefinition* – описывает атрибут, которым могут обладать объекты каких-либо классов модели.

Атрибуты тега AttributeDefinition:

AttributeId – уникальный код атрибута

Name – читаемое наименование атрибута

Type – тип значения атрибута: Literal для атрибутов-литералов, Reference для атрибутов, хранящих ссылки на другие объекты.

DataType – тип значения для атрибутов-литералов, один из стандартных типов данных xsd.

MinCardinality – минимальное число значений этого атрибута для каждого объекта. Если MinCardinality=1, значит, атрибут обязателен.

MaxCardinality – максимальное число значений этого атрибута для каждого объекта.

Archive – значение true соответствует устаревшим атрибутам модели.

Тег *Target*

Указывает классы, объекты которых могут быть значениями для данного атрибута-ссылки.

Атрибуты тега Target

TargetID – уникальный код класса

Name – читаемое наименование класса

Тег *ApplicableAttribute*

Передает информацию о том, что данный атрибут присущ объектам описываемого класса.

Атрибуты тега ApplicableAttribute

AttributeID – уникальный код атрибута.

Пример ответа

В формате XML:

```
<DataSchemaCompact Destination="test" Prefix="http://trinidata.ru/demo/"
StartElement="СправочникТиповыеРешения">
  <AttributeDefinition AttributeId="ОтноситсяКДисциплине" Name="Относится к
дисциплине" Archive="false" Type="Reference" >
    <Target TargetId="СправочникДисциплина" Name="Дисциплина" />
  </AttributeDefinition>
```

```
  <AttributeDefinition AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Name="Наименование" Type="Literal" DataType="xsd:string"
Archive="false" MaxCardinality="1" />
  <ObjectType Code="СправочникТиповыеРешения" Name="Типовые решения"
Archive="false" >
    <Parent ParentId="Справочники" />
    <ApplicableAttribute AttributeId="ОтноситсяКДисциплине" />
    <ApplicableAttribute AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" />
  </ObjectType>
</DataSchemaCompact>
```

4.3.3. Получение информационной модели в форматах OWL, N-Triples и Turtle

Информационная модель может так же быть получена в одном из трех стандартных форматов:

- **DataModelOwlRequest** - в формате OWL;
- **DataModelNtriplesRequest** – в формате N-Triples;
- **DataModelTurtleRequest** – в формате Turtle.

Параметры

Все три метода могут быть вызваны с указанием с флага **Original**. Атрибут не обязателен, принимает значения 0 и 1, значение по умолчанию 0. Если в запросе передано значение Original равное 1, ответ содержит только модель в требуемом формате. В противном случае будут возвращены пакеты **DataModelOwl**, **DataModelNTriples** и **DataModelTurtle** соответственно, содержащие модель в качестве значения атрибута **Result**.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataModelOwlRequest Endpoint="demo" Originator="test" />
```

Пример ответа

В формате XML:

```
<DataModelOwl Endpoint="demo" Destination="test" Result="&lt;rdf:RDF&#10;
xmlns:rdf=&quot;http://www.w3.org/1999/02/22-rdf-syntax-ns#&quot;&#10;
xmlns:owl=&quot;http://www.w3.org/2002/07/owl#&quot;&#10;
…
&lt;rdf:type
rdf:resource=&quot;http://trinidata.ru/demo/Персона&quot;/&gt;&#10;
&lt;/owl:NamedIndividual&gt;&#10;&lt;/rdf:RDF&gt;&#10;"/>
```

Пример запрос с указанием атрибута Original:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataModelOwlRequest Endpoint="demo" Originator="test" Original="1"/>
```

Ответ:

```
<rdf:RDF
```

```
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:j.0="http://trinidata.ru/demo/"
        xmlns:j.1="http://www.w3.org/ns/shacl#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:j.2="http://www.w3.org/ns/org#"
        xmlns:j.3="http://www.onto.pro/"
        xmlns:j.4="http://trinidata.ru/archigraph-mdm/">
…
        <rdf:type rdf:resource="http://trinidata.ru/demo/Персона"/>
    </owl:NamedIndividual>
</rdf:RDF>
```

В случае, если запрашивающая система не имеет доступа ко всем классам информационной модели хотя бы на чтение, получение описания модели указанными методами будет не успешно. В этом случае в качестве ответа запрашивающая система получит пакет InvalidPackage с сообщением о недостаточных правах доступа.

4.4. Запросы на получение информации об объектах

4.4.1. Получение информации об одном объекте

GetObject – запрос на получение конкретного объекта

Обязательные параметры

Обязательный набор параметров содержит только **Code** – код (идентификатор АрхиГраф) объекта, описание которого необходимо получить.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObject Originator="test" Code="ИвановИИ" />
```

В формате JSON:

```
{"GetObject": {"Originator": "test", "Code": "ИвановИИ"}}
```

Ответ

Items – пакет с описанием объектов. Подробное описание пакета приведено ниже в описании запроса на получение набора объектов.

Прочие параметры

Также в запросе GetObject могут быть заданы ограничения на атрибутный состав возвращаемого объекта и/или требование вернуть дополнительную информацию (на права доступа к объекту; объекты, на которые ссылается запрошенный объект; хранилище, в котором лежит или должен лежать объект и т.п.).

Приведем перечень атрибутов и вложенных тегов, применимых к запросу GetObject (подробное описание и примеры будут приведены далее):

- **WithHistory** – получить историю изменения объекта;
- **Date** - получить состояние объекта на указанную дату;
- **SourceSystem** - фильтрация по системе-источнику (для специальных хранилищ);
- **ReturnStorage** - получить информацию о хранилищах объекта;
- **ReturnRights** - получить права доступа к объекту;
- **ReturnCodeOnly** – получить только идентификатор объекта;
- **ReturnTypeOnly** – получить по объекту его идентификатор и принадлежности классам модели;
- **WithoutName** – для значений атрибутов ссылочного типа не возвращать читаемые наименования;
- **ReturnAllTypes** – вернуть с объектом все классы модели, к которым он принадлежит, включая owl:NamedIndividual;
- **Check** – применить к объекту правила контроля логической целостности;
- **UsePlace** – искать объект только в хранилищах с указанным размещением;
- **AllValues** – вернуть в том числе значения всех заданных у объекта атрибутов, в том числе устаревших;
- **ReturnLinkedObjects** – вернуть связанные объекты.
- Тег **FieldSet** – задать набор атрибутов, возвращаемых для объекта.

Флаг **WithHistory** используется, если вместе с описанием объекта требуется вернуть историю изменения его свойств. Запрос GetObject, содержащий только код объекта и атрибут WithHistory=1 идентичен запросу GetObjectHistory (см. раздел 5.2.1).

Атрибут **Date** типа дата и время позволяет получить состояние объекта на указанную дату. Подробнее см. раздел 5.2.2.

Для некоторых реализаций хранилищ может иметь смысл дополнительная фильтрация по системе-источнику. Для передачи этого ограничения предназначен строковый необязательный атрибут **SourceSystem**. В базовой версии хранилищ, использующих разделения по системе источнику, нет, но оно может возникнуть при реализации специализированных адаптеров с внешними источниками данных.

Пример:
```
<GetObject
  ReturnStorage="1"
  Code="ТестовыйКлассPostgres"
/>
```

Атрибут **ReturnStorage** принимает значения 0 и 1 (по умолчанию 0) и позволяет получить информацию о физическом размещении объекта. В случае, если в запросе передано ReturnStorage=1, ответ будет дополнен тегом Storages, содержащим следующие вложенные теги:
- **Model** - в каком хранилище или хранилищах, согласно модели, могут содержаться объекты тех классов, к которым принадлежит запрошенный объект;
- **Actual** – в каком хранилище фактически лежат данные об объекте;
- **Objects** – возвращается только в том в случае, если объект является классом модели. Тогда в теге (тегах) Objects содержится информация о месте хранения (согласно настройкам модели) объектов, принадлежащих запрошенному классу.

Описание хранилища содержит атрибуты:
Name – читаемое наименование хранилища;
PhpClass – php-класс MDM, реализующий работу с указанных хранилищем;
Host, **Port**, **Dbname**, **Table** – реквизиты доступа к хранилищу;
Editable – флаг (значения 0 или 10 – является ли хранилище редактируемым;
Multilang – флаг, поддерживает ли хранилище мультиязычные данные.

Хранилища с именами вида «Default owl:[Class, NamedIndividual, …] storage» относятся к внутренним служебным хранилищам самой MDM. Управление данными в них происходит через настройки MDM.

Пример запроса с использованием ReturnStorage:

```
<GetObject ReturnStorage="1"Code="ТестовыйКлассPostgres" />
```

Ответ:

```
<Items Count="1">
  <Item Code="ТестовыйКлассPostgres">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Класс"/>
    . . .
    <Storages>
      <Model PhpClass="mdm_storage_default" Name="Default owl:Class storage"
Editable="0" Multilang="1"/>
      <Model PhpClass="mdm_storage_fuseki" Name="Fuseki Demo" Host="127.0.0.1"
Port="8080" Table="demo" Editable="1" Multilang="1"/>
      <Actual PhpClass="mdm_storage_fuseki" Name="Fuseki Demo" Host="127.0.0.1"
Port="8080" Table="demo" Editable="1" Multilang="1"/>
       <Objects PhpClass="mdm_storage_postgresql" Name="Тестовое хранилище
PostgreSQL" Host="127.0.0.1" Dbname="mdm_data" Table="test_storage" Editable="1"
Multilang="0"/>
     </Storages>
  </Item>
</Items>
```

Флаг **ReturnRights** предназначен для того, чтобы получить права доступа к объекту для запрашивающей системы. Уровень прав доступа будет возвращен в атрибуте **Access** тега Item. Возможные значения атрибута Access следующие:

- *readOnly* – система имеет права доступа к объекту только на чтение;
- *editableWithConfirmation* – система имеет права на редактирование с подтверждением;
- *editable* - система имеет полные права к объекту, в том числе на редактирование и удаление.

Пример использования ReturnRights:

Запрос:

```
<GetObject Endpoint="demo" Originator="test" Code="Персона" ReturnRights="1"/>
```

```
Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="Персона" Name="Персона" Access="editable">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Класс"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Персона"/>
    …
  </Item>
</Items>
```

Если необходимо только проверить существование объекта, запрос можно выполнить с указанием флага **ReturnCodeOnly**=1. В этом случае по объекту вернется только его идентификатор в том случае, если объект существует, и сообщение об ошибке, сели объекта с таким кодом нет.

Пример для флага ReturnCodeOnly:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="Персона" ReturnCodeOnly="1"/>

Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="Персона"/>
</Items>
```

При передаче в запросе флага **ReturnTypeOnly**=1 для объекта будет возвращены только его код и информация о принадлежности объекта классам модели.

Пример для флага ReturnTypeOnly:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" ReturnTypeOnly="1"/>

Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ">
    <Type TypeId="Персона" Name="Персона"/>
  </Item>
</Items>
```

По умолчанию для значений атрибутов ссылочного типа вместе со значением возвращается его читаемое наименование в атрибуте Name тега Attribute. Флага **WithoutName** позволяет отключить передачу читаемых наименований значений атрибутов-ссылок, если эти наименования не нужным запрашивающей системе. Тем самым сокращается размер передаваемого пакета и время на его формирование.

Пример для флага WithoutName:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" WithoutName="1"/>
```

```
Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
    <Type TypeId="Персона"/>
    <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
  </Item/>
</Items>

Ответ для того же запроса без указания флага WithoutName:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
    <Type TypeId="Персона" Name="Персона"/>
    <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа" Name="ООО
Альфа"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
  </Item>
</Items>
```

Со временем модель данных может изменяться. В случае, если атрибут был удален или изменилась его область применения, по умолчанию не происходит удаления значения данного атрибута у объектов, к которым атрибут был применим ранее. Значения неактуальных атрибутов не будут возвращаться при запросе объекта, но сохранятся в базе и могут быть просмотрены с помощью флага **AllValues**.

Пример для флага AllValues (из модели был ранее исключено свойство Инициалы):

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" AllValues="1"/>

Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
    <Type TypeId="Персона" Name="Персона"/>
    <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа" Name="ООО
Альфа"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
    <Attribute AttributeId="Инициалы" Type="Literal" Value="И.И."/>
  </Item>
</Items>
```

Индивидуальные объекты помимо принадлежности классу модели принадлежат классу owl:Namedindividual. По умолчанию данный класс в описании объекта не возвращается. Если требуется получить по объекту принадлежности всем классам, включая owl:Namedindividual, следует в запросе указать флаг **ReturnAllTypes**=1.

Пример для флага ReturnAllTypes:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" ReturnAllTypes="1"/>
```

```
Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
    <Type TypeId="Персона" Name="Персона"/>
    <Type TypeId="http://www.w3.org/2002/07/owl#NamedIndividual" Name="Объект"/>
    <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа" Name="ООО
Альфа"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
  </Item>
</Items>
```

Платформа АрхиГраф поддерживает правила контроля логической целостности (SHACL Constraints). При запросе объекта можно выполнить проверку того, удовлетворяет ли данный объект определенным в системе правилам. При указании флага **Check**=1 по объекту будет проведена проверка правила контроля логической целостности и в случае, если объект не удовлетворяет какому-либо из правил, в ответе в атрибуте Message вернется описание нарушения. Также будет создан объект класса shacl:ValidationResult, связанный с проверенным объектом и описывающий нарушение.

Пример для флага Check:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" Check="1"/>

Ответ:
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И." Message="Для персоны не задан ИНН"/>
    <Type TypeId="Персона" Name="Персона"/>
    <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа" Name="ООО
Альфа"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
  </Item>
</Items>
```

Для хранилища объектов может быть задано значение свойства Place – площадка. Чтобы поиск объектов происходил только среди хранилищ заданной площадки, в запросе GetObject можно указать название этой площадке в атрибуте **UsePlace**.

Атрибут **ReturnLinkedObjects** принимает целочисленные положительные значения и позволяет вернуть вместе с запрошенным те объекты, на которые указывают атрибуты ссылочного типа. Значение атрибута – глубина построения связей. В случае, если передано ненулевое значение атрибута ReturnLinkedObjects, в корневом тебе ответа наряду с атрибутом Count, содержащем число найденных объектов, будет возвращен атрибут **LinkedObjectsCount** – число связанных объектов. У связанных объектов тег Item дополняется атрибутом **LinkedObject** – глубиной связи относительно запрошенного объекта.

Пример для флага ReturnLinkedObjects:

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ" ReturnLinkedObjects
="2"/>
```

Ответ:
```
<Items Count="1" LinkedObjectsCount="2" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
      <Type TypeId="Персона" Name="Персона"/>
      <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Альфа" Name="ООО
Альфа"/>
      <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
   </Item>
   <Item Code="Альфа" Name="ООО Альфа" LinkedObject="1">
      <Type TypeId="Организация" Name="Организация"/>
      <Attribute AttributeId="УчаствуетВПроекте" Type="Reference" Value="Проект_1"
Name="Тестовый проект"/>
      <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="ООО Альфа"/>
   </Item>
   <Item Code="Проект_1" Name="Тестовый проект" LinkedObject="2">
      <Type TypeId="Проект" Name="Проект"/>
      <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Тестовый проект"/>
   </Item>
</Items>
```

Примечание: если объект запрошен с ограничением на атрибутный состав, например, в запросе указаны флаги ReturnCodeOnly, ReturnTypeOnly или тег FieldSet, то ReturnLinkedObjects будет влиять на поиск связанных объектов не по всем свойствам объекта, а только по тем, которые содержатся в ответе. В частности, так как при запросе объекта с флагами ReturnCodeOnly и ReturnTypeOnly атрибуты не возвращаются, то поиск связанных объектов смысла не имеет.

Тег **FieldSet** позволяет задать набор свойств, которые требуется вернуть по объекту. Атрибут **Exclude** тега FieldSet принимает значения 0 и 1 (значение по умолчанию 0) указывает, следует ли указанные свойства включать в ответ (Exclude=0) или исключать из него (Exclude=1). Идентификаторы свойств, которые требуется включить или исключить из ответа, указываются в атрибуте **AttributeId** тегов **Field**, вложенных в тег FieldSet.
Пример 1: вернуть в ответе только читаемое наименование объекта

Запрос:
```
<GetObject Endpoint="demo" Originator="test" Code="ИвановИИ">
  <FieldSet>
    <Field AttributeId="http://www.w3.org/2000/01/rdf-schema#label"/>
  </FieldSet>
</GetObject>
```

Ответ:
```
<Items Count="1" Endpoint="demo" Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
      <Type TypeId="Персона" Name="Персона"/>
      <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Иванов И.И."/>
   </Item>
</Items>
```

Пример 2: вернуть для объекта все свойства, кроме координат центра

```
Запрос:
<GetObject Endpoint="demo" Originator="test" Code="ПлощадноОбъект_1">
  <FieldSet Exclude="1">
    <Field AttributeId="Широта"/>
    <Field AttributeId="Долгота"/>
  </FieldSet>
</GetObject>

Тот же запрос в формате JSON:

{"GetObject": {
   "Endpoint":"demo", "Originator":"test", "Code":"ПлощадноОбъект_1",
   "FieldSet": [
     {
        "Exclude":"1",
        "Field": [
           {"AttributeId":"Широта"},
           {"AttributeId":"Долгота"}
        ]
     }
   ]
 }}
```

4.4.2. Получение информации о наборе объектов

GetObjectsGroup – запрос на получение всех объектов определенного класса или классов, удовлетворяющих заданным условиям фильтра.

Параметры

Запрос имеет два формата. В сокращенном формате запрос состоит из одного тега GetObjectsGroup, в атрибуте Code которого указывается конкретный тип объектов, который необходимо вернуть. Возвращаются все объекты, относящиеся к данному классу.

В полном формате тег GetObjectsGroup включает несколько вложенных тегов ObjectType, что позволяет выбрать экземпляры, относящиеся к нескольким классам (хотя бы к одному из перечисленных). Теги FilterGroup и Filter позволяют задать условия отбора. Описание их синтаксиса приведено ниже.

Тег *GetObjectsGroup*

Корневой тег запроса.

Атрибуты:

Code – идентификатор класса, объекты которого необходимо вернуть (при сокращенном формате запроса).

Все остальные теги и атрибуты используются только в полном формате запроса.

Атрибуты корневого тега:

ObjectTypeGroupOperation — может принимать значение and или or. Используется для обработки условий принадлежности объектов классам, задаваемых в тегах ObjectType.

Атрибут не является обязательным; если он не указан, то это равносильно заданию значения or (логическое ИЛИ).

CombineGroups – необязательный атрибут, принимает значения and или or. Используется для объединения в группу несколько условий отбора, заданных тегами FilterGroup. Если атрибут не указан, это равносильно значению and (логическое И).

CombineGeometry – необязательный атрибут, принимает значения and или or. Используется для объединения в группу несколько условий отбора по координатам, заданных тегами Geometry. Если атрибут не указан, это равносильно значению or (логическое ИЛИ).

ReturnCount – необязательный атрибут, принимает значения 0 и 1, по умолчанию значение равно 0. В случае переданного значения атрибута 1 в ответном пакете вместо информации об объектах будет возвращено их количество (в атрибуте Count тега Items).

WithoutSubClasses – необязательный атрибут со значениями 0 и 1, по умолчанию считается равным 0. Если в атрибуте WithoutSubClasses передано значение 1, то в условиях отбора будут учитываться только объекты, принадлежащие непосредственно заданным классам. Если атрибут WithoutSubClasses равен 0 или не задан, то в формировании условия на отбор объектов учитываются так же все подклассы классов, переданных в тегах ObjectType и/ атрибуте Code тега GetObjectsGroup.

WithoutRefChecking – необязательный атрибут, принимает значения 0 и 1, по умолчанию равен 0. Позволяет отключить проверку существования объектов в фильтрах по атрибутам ссылочного типа. Данная опция с одной стороны позволяет сократить время выполнения запроса, но с другой стороны в случае, если фильтр задан некорректно, запрашивающая система не получит информацию об этом.

Limit – целое число: ограничение на количество возвращаемых объектов. Не обязательный атрибут, значение по умолчанию 1000.

Offset – необязательный атрибут, по умолчанию равен 0. Целое число, смещение: количество пропускаемых в результате поиска объектов. Найденные объекты нумеруются с 0.

Следующая группа атрибутов тега GetObjectsGroup идентична атрибутам запроса getObject, и их описание и примеры приведены в предыдущем разделе:

ReturnRights – вернуть для каждого найденного объекта права доступа запрашивающей системы к объекту;
SourceSystem - фильтрация по системе-источнику;
ReturnCodeOnly – вернуть только идентификаторы найденных объектов;
ReturnTypeOnly – для найденных объектов вернуть их идентификаторы и принадлежности классам модели;
WithoutName – для значений атрибутов ссылочного типа не возвращать читаемые наименования;
ReturnAllTypes – для найденных объектов вернуть все классы модели, к которым он принадлежит, включая owl:NamedIndividual;
Check – применить к найденным объектам правила контроля логической целостности;
UsePlace – искать объекты только в хранилищах заданной площадки;
AllValues – вернуть по объектам в том числе значения неактуальных атрибутов;

ReturnLinkedObjects – вернуть связанные объекты.

Тег *ObjectType*

Позволяет указать классы, к которым могут относиться результаты запроса.

Атрибуты:

Code – идентификатор класса, объекты которого необходимо вернуть.

В составе запроса может присутствовать несколько тегов ObjectType с разными значениями Code. Возвращаются объекты, относящиеся в зависимости от значения атрибута ObjectTypeGroupOperation тега GetObjectsGroup либо к хотя бы к одному из перечисленных классов (при ObjectTypeGroupOperation=«or», или если значение атрибута ObjectTypeGroupOperation не указано) либо ко всем перечисленным классам (если значение ObjectTypeGroupOperation равно «and»). При этом учитывается вложенность классов: объекты, принадлежащие подклассам запрошенных классов, тоже возвращаются (если не передан атрибут WithoutSubClasses=1 в корневом теге запроса).

Тег *FilterGroup*

Объединяет в группу несколько условий отбора результатов запроса. За логику объединения отвечает атрибут CombineGroups корневого тега GetObjectsGroup. По умолчанию группы объединены логическим И.

Атрибуты тега FilterGroup

Operation – логическая операция, которой объединяются условия фильтра, заданные внутри группы. Принимает значения and и or.

Тег *Filter*

Описывает одно из условий отбора результатов запроса.

Атрибуты тега Filter

Attribute – идентификатора атрибута, к которому применяется условие.

Value – литерал, с которым сравнивается значение указанного атрибута у всех отбираемых результатов. Если атрибут является указателем на другой объект, то Value должен содержать идентификатор объекта.

Comparison – операция сравнения. Принимаемые значения:

- Equal (равно),
- Contains (содержит),
- More (больше),
- Less (меньше),
- MoreOrEqual (больше или равно),
- LessOrEqual (меньше или равно),
- NotEqual (не равно),
- Exists (значение задано),
- NotExists (значение атрибута не задано),

- iEqual (равно с точностью до регистра).

Не все операции сравнения применимы ко всем атрибутам. Например, к атрибутам, принимающих числовые значение, а также значения типов дата и дата и время, неприменимы операции Contains и iEqual. К атрибутам булевского типа применимы только операции Equal/ NotEqual и Exists/NotExists. Для строковых атрибутов и атрибутов ссылочного типа не могут указываться операции больше, больше или равно, меньше, меньше или равно. В случае, если в условии фильтрации задана операция сравнения, не применимая к данному атрибуту, система вернет сообщение об ошибке.

Variable – необязательный атрибут, содержит имя переменной, участвующей в условии (для многоуровневых запросов). Использование в фильтрах переменных позволяет строить достаточно сложные условия отбора, учитывающие цепочки связей, и налагать условия не только на свойства самого объекта, но на свойства связанных с ним объектов.

ByName – необязательный атрибут, принимает значения 0 и 1, значение по умолчанию - 0. Имеет смысл только для атрибутов-указателей. В случае, если ByName равно 1, в условии отбора для сравнения с Value используется не значение атрибута, а читаемое наименование объекта, на который атрибут указывает.

Тег *Geometry*

Позволяет вести поиск объектов по географическим координатам, сохраненным в формате GeoJSON. Применим не для всех данных, а только тех, хранилище которых допускает такого рода поиск. В случае, если хранилище не поддерживает геопоиск, условие всегда будет ложно. С остальными условиями ограничение на координаты объединяется логическим И. Если задано несколько тегов Geometry, то между собой они будут объединены в зависимости от значения атрибута CombineGeometry головного тега GetObjectsGroup – логическое И либо ИЛИ, по умолчанию – ИЛИ.

Атрибуты тега Geometry:

Type – тип географического объекта: точка(point), полигон (polygon).

Attribute – атрибут, хранящий координаты

Тег *Coordinates*

Задает координаты точек – границ полигона в поиске по GeoJSON

Атрибуты тега Coordinates

Longitude - географическая долгота – вещественное число из диапазона (-180,180]

Latitude – географическая широта – вещественное число из диапазона (-90,90]

Тег *Item*

Позволяет передать идентификаторы искомых объектов. Условия тега Item объединяются логическим И с остальными условиями: если какие-то из объектов с переданными идентификаторами не удовлетворяют ограничениям, заданным с помощью тегов ObjectType, FilterGroup, Geometry и атрибутов головного тега, то в ответе данные объекты возвращены не будут.

Атрибуты тега Item

Code – идентификатор объекта
Variable – имя переменной, участвующей в условии (для запросов с подзапросами). Должен быть обязательно задан один двух этих атрибутов Code или Variable.
Comparison – операция сравнения. Принимаемые значения:

- Equal (равно),
- NotEqual (не равно),
- iEqual (равно с точностью до регистра).

Если атрибут Comparison не задан, то он полагается равным значению по умолчанию Equal (равно).

Пример запроса с использованием тега Item: среди заданных идентификаторами сотрудников вернуть только тех, кто работает в компании Альфа:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Сотрудники"/>
  <FilterGroup Operation="And">
    <Filter Attribute="РаботаетВ" Value="Альфа" Comparison="Equal" />
  </FilterGroup>
  <Item Code="ИвановИИ" />
  <Item Code="ПетровПП" />
  <Item Code="СидоровСС" />
</GetObjectsGroup>
```

Пример запроса с использованием тега Item в формате json:

```
{"GetObjectsGroup": {
    "ObjectType":[
    {"Code":"Сотрудник"}
  ],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"РаботаетВ",
             "Value":"Альфа",
             "Comparison":"Equal"
           }
     ]
   }
    ],
    "Item": [
    {"Code":"ИвановИИ"},
    {"Code":"ПетровПП"},
    {"Code":"СидоровСС"}
    ]
  }
}
```

Пример запроса с атрибутом Comparison: вернуть всех сотрудников Альфа кроме указанного:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Сотрудники"/>
  <FilterGroup Operation="And">
```

```
    <Filter Attribute="РаботаетВ" Value="Альфа" Comparison="Equal" />
  </FilterGroup>
  <Item Code="ИвановИИ" Comparison="NotEqual" />
</GetObjectsGroup>
```

Пример запроса с использованием Variable будет приведен далее при описании подзапросов.

Тег *Sort*

Задает сортировку найденных объектов по указанным атрибутам в порядке возрастания или убывания

Атрибуты тега Sort

AttributeId – идентификатор атрибута, по которому производится сортировка. Обязательный атрибут.

Direction – Направление сортировки – по возрастанию (значение атрибута ASC) либо убыванию (значение DESC). Атрибут не является обязательным, по умолчанию производится сортировка по возрастанию значений атрибута (ASC)

SortByName – необязательный атрибут, принимающий значения 0 и 1. Имеет смысл только при сортировке с по значениям атрибутов-указателей. В случае, если SortByName равно 1, сортировка производится не по значениям атрибута, а по читаемым названиям объектов, на которые ссылается атрибут.

Пример запроса с сортировкой - отсортировать происшествия по наименованию района, в котором оно произошло, а в рамках района – по дате в обратном порядке:

```
<GetObjectsGroup>
  <ObjectType Code="Происшествие"/>
  <FilterGroup Operation="And">
    <Filter Attribute="Дата" Value="2018-06-01" Comparison="More" />
  </FilterGroup>
<Sort AttributeId="ПроизошлоВРайоне" Direction="ASC" SortByName="1" />
<Sort AttributeId="Дата" Direction="DESC"/>
</GetObjectsGroup>
```

Пример запроса с сортировкой в формате JSON:

```
{"GetObjectsGroup": {
    "ObjectType":[{"Code":"Происшествие"}],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"Дата",
             "Value":"2018-06-00",
             "Comparison":"More"
         }
     ]
   }
    ],
  "Sort":[
      {
     "AttributeId":"ПроизошлоВРайоне",
        "Direction":"ASC",
```

```
            "SortByName":"1"
        },
           {
           "AttributeId":"Дата",
           "Direction":"DESC"
        }
      ]
       }
    }
```

Тег *FieldSet*

Позволяет в ответе получить не все атрибуты объекта, а только необходимые.

Атрибуты тега FieldSet:

Exclude – принимает значения 0 и 1, по умолчанию равен 0. Если значение равно 1, то из списка атрибутов объекта в ответе исключаются указанные. Если атрибут равен 0 или не задан, то для объектов будут возвращены значения указанных атрибутов.

Тег *Field* – вложен в FieldSet

Описание атрибутов, которые требуется либо вернуть, либо исключить из ответа в зависимости от значения параметра Exclude.

Параметры тега Field:

AttributeId – идентификатор атрибута

Variable – не обязательный - идентификатор переменной, из которой берется возвращаемое значение. Используется для запросов с подзапросами.

SetVariable – не обязательный, идентификатор переменной, в которую следует положить значение атрибута. Используется для запросов с подзапросами.

Тег *Subrequest* – задает подзапрос

Допускает все те же параметры и теги, которые описаны для тега GetObjectsGroup. Помимо этого, может иметь необязательный атрибут SetVariable – если он задан, то в значение переменной с указанным именем попадут идентификаторы объектов, отобранных условием подзапроса.

Пример запроса

В сокращенной форме в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup Code="Компании" ></GetObjectsGroup>
Пример запроса в сокращенной форме в формате json:
{"GetObjectsGroup":{"Code":"Компании"}}
```

В полной форме в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Компании"/>
  <ObjectType Code="Организации"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="Название" Value="альфа" Comparison="Equal" />
```

```
    <Filter Attribute="Название" Value="бета" Comparison="Contains" />
  </FilterGroup>
</GetObjectsGroup>
```

В полной форме в формате JSON:

```
{"GetObjectsGroup":{
  "Originator":"test",
  "ObjectType":[
    {"Code":"Компании"},
    {"Code":"Организации"}
  ],
  "FilterGroup":[
    {
      "Operation":"Or",
      "Filter":[
      {"Attribute":"Название", "Value":"альфа", "Comparison":"Equal"},
      {"Attribute":"Название", "Value":"бета", "Comparison":"Equal"}
      ]
    }
  ]
 }
}
```

Пример запроса с использованием геометрии

В формате XML:

```
<GetObjectsGroup>
  <ObjectType Code="Происшествие"/>
  <FilterGroup Operation="And">
    <Filter Attribute="Дата" Value="2018-06-00" Comparison="More" />
  </FilterGroup>
   <Geometry Type="Polygon" Attribute="Координаты">
    <Coordinates Longitude="30.3837" Latitude = "59.8768" />
    <Coordinates Longitude="30.3837" Latitude = "59.9168" />
    <Coordinates Longitude="30.4639" Latitude = "59.9168" />
    <Coordinates Longitude="30.4639" Latitude = "59.8768" />
  </Geometry>
</GetObjectsGroup>
```

Поиск по координатам в формате JSON:

```
{"GetObjectsGroup": {
    "ObjectType":[{"Code":"Происшествие"}],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"Дата",
             "Value":"2018-06-00",
             "Comparison":"More"
         }
     ]
   }
    ],
    "Geometry": [
    {
      "Type":"Polygon",
      "Attribute":"Координаты",
          "Coordinates":[
            {"Longitude":"30.3837", "Latitude":"59.8768"},
```

```
            {"Longitude":"30.3837", "Latitude":"59.9168"},
            {"Longitude":"30.4839", "Latitude":"59.9168"},
            {"Longitude":"30.4839", "Latitude":"59.8768"}
          ]
    }
    ]
  }
}
```

Пример запроса с использованием FieldSet

В формате XML – вернуть для объектов класса Парк только значение атрибута Территория:

```
<GetObjectsGroup Code="Парк" Limit="20" Offset="0">
  <FieldSet Exclude="0">
        <Field AttributeId="Территория" />
  </FieldSet>
</GetObjectsGroup>
```

В формате JSON:

```
{"GetObjectsGroup": {
  "Code":"Парк",
  "Limit":"20",
  "Offset":"0",
  "FieldSet": [
    {
    "Exclude":"0",
        "Field": [
          {"AttributeId":"Территория"}
     ]
   }
    ]
  }
}
```

Пример запроса с подзапросом: найти всех сотрудников действующих компаний с фамилией Иванов. В ответе получить для каждого сотрудника его ФИО и название компании, в которой он работает:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Сотрудник"/>
  <FilterGroup Operation="And">
    <Filter Attribute="ФИО" Value="Иванов" Comparison="Contains" />
    <Filter Attribute="РаботаетВ" Variable="?x" Comparison="Equal" />
  </FilterGroup>
  <Subrequest SetVariable="?x">
    <ObjectType Code="Компания"/>
    <FilterGroup Operation="And">
     <Filter Attribute="Действующая" Value="true" Comparison="Equal" />
    </FilterGroup>
    <FieldSet>
      <Field AttributeId="ПолноеНаименование" SetVariable="?y" />
   </FieldSet>
  </Subrequest>
  <FieldSet>
    <Field AttributeId="ФИО" />
    <Field AttributeId="РаботаетВ" Variable="?y" />
  </FieldSet>
  <Sort AttributeId="ФИО" Direction="ASC" />
</GetObjectsGroup>
```

Запрос может содержать несколько подзапросов. В том числе допускается ситуация, когда значение переменной одного подзапроса используется в качестве значения другого подзапроса.

Пример: найти все Магазины, принадлежащие Индивидуальным предпринимателям, отобранным по условиям. При этом принадлежность Магазина ИП задана через объект-связь ОтношениеКТорговомуОбъекту.

```
<GetObjectsGroup>
  <ObjectType Code="Магазин" />
   <Item Variable="?shop" />
   <Subrequest SetVariable="?ip">
    <ObjectType Code="ИндивидуальныйПредприниматель"/>
     <FilterGroup Operation="and">
      <Filter Attribute="rdfs:label" Value="Иванов " Comparison="contains" />
        <Filter Attribute="ИНН" Value="6658" Comparison="Contains" />
     </FilterGroup>
    </Subrequest>
    <Subrequest>
      <ObjectType Code="ОтношениеКТорговомуОбъекту" />
      <FilterGroup Operation="and">
        <Filter Attribute="являетсяСубъектомРаспорядителем" Variable="?ip"
Comparison="Equal" />
      </FilterGroup>
      <FieldSet>
        <Field AttributeId="отношениеОпределеноДляМагазина" SetVariable="?shop" />
      </FieldSet>
    </Subrequest>
</GetObjectsGroup>
```

В результате выполнения данного запроса будет выполнен сперва первый подзапрос и найдены все ИП с фамилией Иванов и ИНН, содержащим указанную подстроку. Далее идентификаторы найденных объектов будут использованы во втором запросе для поиска идентификаторов связанных с ИП магазинов. И наконец основной запрос найдет полное описание магазинов, определенных подзапросами.

Ответ

Пакет **Items**

Пакет Items содержит информацию о конкретных объектах. Данный пакет может быть получен в качестве результата выполнения запросов GetObject, GetObjectsGroup, GetObjectHistory.

Пакет состоит из одного или нескольких элементов Item, каждый из которых передает информацию об одном объекте. В рамках одного пакета Items может прийти информация о нескольких взаимосвязанных сущностях разных типов, или о нескольких сущностях одного типа.

Каждый объект (сущность) характеризуется следующими параметрами:

- **Идентификатор** – глобальный код объекта, присвоенный системой АрхиГраф. Обычно выглядит как URI, то есть адрес вида http://some-domain.ru/prefix/object.

Первая часть идентификатора, до object, называется префиксом - обычно она одинакова для всех элементов модели. Стандартный префикс может указываться, или пропускаться. Префикс, отличный от стандартного, указывается в любом случае. Последняя часть URI, object, является собственно уникальным кодом объекта.

- **Наименование** – стандартное свойство, присутствующее по умолчанию у всех элементов. Представляет собой читаемое название элемента.
- Набором **типов** – перечнем классов, к которым относится объект. Каждый объект может относиться к любому количеству классов. Принадлежность к классам рекурсивна, т. е. для АрхиГраф тот факт, что объект принадлежит к некому классу иерархии, означает, что он одновременно является членом всех вышележащих классов.
- Набором **атрибутов** – перечнем значений свойств данного объекта. Для свойств-литералов указывается непосредственное значение, для свойств-связей - уникальный идентификатор объекта, на которое ссылается свойство. В семантической модели каждое свойство каждого объекта может иметь столько значений, сколько разрешает информационная модель.

Пример пакета, иллюстрирующий его общую структуру:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И.">
   <Type TypeId="Персона" Name="Персоны" />
   <Type TypeId="Сотрудник" Name="Сотрудники" />
   <Attribute Type="Literal" AttributeId="ДатаРождения" Value="1979-01-01"/>
   <Attribute Type="Reference" AttributeId="РаботаетВ" Value="ООООльфа"/>
  </Item>
</Items>
```

Пример пакета Items в формате JSON:

```
{"Items": {
  "Destination":"test",
  "Item": [
    {
    "Code":"ИвановИИ",
    "Name":"Иванов И.И.",
    "Type": [
      {"TypeId":"Персона", "Name":"Персоны"},
      {"TypeId":"Сотрудник", "Name":"Сотрудники"}
    ],
    "Attribute": [
      {
        "Type":"Literal",
        "AttributeId":"ДатаРождения",
         "Value":"1979-01-01"
      },
      {
        "Type":"Reference",
        "AttributeId":"РаботаетВ",
        "Value":"ООООльфа"
       }
    ]
  }
    ]
```

```
    }
}
```

Тег *Item*

Передает информацию о конкретном объекте. Тег Item может быть либо возвращен в составе пакетов Items, SubscriptionItems, SubscriptionDeleteItems, либо получен в составе входящего пакета UpdateObject, содержащего запрос на редактирование/создание объекта. Ниже приводится описание атрибутов тега Item с уточнением, в каком случае он может указан.

Атрибуты тега Item

Code – уникальный код объекта

Name – читаемое имя объекта

LocalCode – временный код объекта, используемый в случае, когда тег Item передается в составе пакета UpdateObject, содержащего запрос на создание объекта. Значением атрибута является внутренний код объекта в той системе, в которой он был создан. Атрибут Code при этом передается пустым. В ответном пакете OperationResults система возвращает присвоенный объекту постоянный уникальный код, а также соответствие между ним и временным кодом.

OperationId – уникальный идентификатор операции на создание/изменение объекта. Используется при передаче тега Item в составе пакета UpdateObject. Генерируется приложением-отправителем, используется для получения информации о результате операции.

DomainIntersection – необязательный атрибут. Может использоваться только при передаче в Item данных о свойствах классов (литеральных и ссылочных). Значение DomainIntersection=true указывается, если областью применимости свойства является пересечение набора классов. По умолчанию в случае задания у свойства нескольких классов в атрибуте domain областью применимости свойства будет объединение этих классов; DomainIntersection в теге Item в этом случае не указывается.

RangeIntersection – необязательный атрибут, так же относится только к передаче данных о свойствах. RangeIntersection=true задается, если областью значений свойства-ссылки является пересечение классов; в случае области значений -объединения классов RangeIntersection не указывается.

IgnoreTypes – необязательный атрибут используется при передаче тега Item в составе пакета UpdateObject, принимает значения 0 или 1, значение по умолчанию 0. При указании атрибута равным 1 принадлежности объекта классам в тегах Type внутри тега Item можно не указывать; если же теги Type будут заданы, то их значения при обработке запросы будут проигнорированы - сохранится существующая принадлежность объекта классам. Имеет смысл только для случая редактирования объекта. Запрос на создание объекта с указанием IgnoreTypes=1 вернет сообщение об ошибке.

AddTypes – необязательный атрибут используется при передаче тега Item в составе пакета UpdateObject, принимает значения 0 или 1, значение по умолчанию 0. При задании AddTypes=1 классы, переданные в тегах Type будут добавлены к классам

объекта (если они были заданы ранее). При AddTypes=0 и без указания IgnoreTypes=1 (по умолчанию) набор классов, которым принадлежит объект, будет заменен на набор классов, переданных в тегах Type.

CreateIfNotExists – необязательный атрибут, используется при передаче тега в составе UpdateObject. Если для тега Item передано значение атрибута CreateIfNotExists=1 и задан атрибут Code, но в системе нет атрибута с указанным идентификатором, то будет создан новый объект с идентификатором, значение которого равно значению атрибута Code. Значение CreateIfNotExists по умолчанию 0 - тогда, если объект с идентификатором из атрибута Code в системе отсутствует, будет возвращено сообщение об ошибке.

Prefix – необязательный строкой атрибут - префикс, используемый для генерации идентификатора нового объекта.
При создании объекта с указанием LocalCode и без задания флага CreateIfNotExists равным 1, идентификатор нового объект генерируется автоматически по определенным правилам:
- при создании классов модели (объекты с типом owl:Class) если для него задано читаемое наименование (свойство rdfs:label), то читаемое наименование разбивается на слова (блоки, состоящие из букв и цифр), начальная буква слова приводится к верхнему регистру и в качестве идентификатора берется конкатенация эти слов. Пример: для наименования «Тестовый класс 1» будет сгенерирован идентификатор «ТестовыйКласс1».
- при создании свойств (owl:DatatypeProperty) и свойств-ссылок (owl:ObjectProperty) идентификатор формируется по аналогичному правилу, за исключением того, что первый символ первого слова приводится к нижнему регистру. Пример: по наименованию «Тестовое свойство-ссылка» будет сгенерирован идентификатор «тестовоеСвойствоСсылка».
- при создании индивидуальных объектов идентификатор генерируется из идентификатора класса объекта (если классов несколько, то первого в списке) в качестве префикса плюс случайная часть, объединенная с префиксом через символ нижнее подчеркивание. Например, при создании объекта класса Персона будет сгенерирован идентификатор вида «Персона_875d6691b703f3015a42c95f245bc6a0».
Если для класса или свойства не задано читаемое наименование, то идентификатор будет сгенерирован по тому же принципу, что и для индивидуального объекта.
В случае, если для объекта в качестве префикса требуется использовать значение, отличное от класса объекта, этот префикс указывается в атрибуте Prefix.

FullUpdate - необязательный атрибут, используется при передаче тега в составе UpdateObject. Принимает значение 0 и 1, значение по умолчанию 0. Если атрибут FullUpdate не задан или равен 0, то изменения затронут только те атрибуты, которые были переданы запросом. В противном случае объект будет обновлен полностью: у объекта будут удалены все существующие атрибуты, которые не переданы в запросе, за исключением атрибутов система-источник и код в системе источнике.

Date - тип дата и время - возвращается в случае, если Item содержит ответ на запрос получения состояния объекта на указанную дату (GetObject или GetObjectsGroup с дополнительным параметром Date).

Тег *Type* – вложен в Item

Передает информацию о том, каким классам принадлежит данный объект. Может встречаться несколько раз для одного объекта.

Атрибуты тега Type

TypeId – идентификатор класса, которому принадлежит объект.

Name – читаемое наименование класса. Имеет смысла только для исходящих пакетов. В случае, если значение атрибута Name передано в рамках запроса UpdateObject, оно будет проигнорировано.

Теги *Operation* – вложен в Item

Возвращается в случае получения ответа на запрос GetObjectHistory и содержит информацию об истории изменения принадлежности класса объекта.

Атрибуты тега Operation:

Date – тип дата и время – момент установки указанного значения

System – код информационной системы, произведшей изменение значения

OperationId – идентификатор операции, в результате которой было изменено значение (если был указан в запросе на изменение)

Plan – возвращается значение 0 для фактически выполненных изменений, значение 1 для изменений, запланированных на будущую дату.

Вложенные теги для тега Operation: Set (один, несколько или ни одного в случае, если значение атрибута было очищено). Единственный атрибут тега Set – Value – содержит установленное значение.

Тег *Attribute* – вложен в Item

Передает информацию о значении какого-либо атрибута объекта. Может встречаться несколько раз для одного атрибута, если модель данных это позволяет.

Атрибуты тега Attribute

Type – тип значения атрибута: Literal, если значение представляет собой константу, Reference – если значение является уникальным идентификатором другого объекта, и LocalCodeReference, если ссылка осуществляется на другой объект по его временному коду. Последний вариант используется только в случаях, когда тег Item передается в составе пакета UpdateObject. При этом пакет может содержать запрос на создание сразу нескольких сущностей, связанных между собой. Значение LocalCodeReference используется для указания на то, что значение Value ссылается на другой объект, создаваемый в этом же запросе, по его временному коду.

AttributeId – идентификатор атрибута в модели данных.

Value – значение атрибута.

Name – читаемое наименование объекта, идентификатор которого содержится в значении атрибута. Возвращается только для атрибутов ссылочного типа. Будет проигнорировано в случае, если значение атрибута Name передано в рамках запроса UpdateObject.

Ignore – используется при передаче тега Item в составе пакета UpdateObject, и указывает, что система не должна изменять значение этого атрибута.

Empty – используется при передаче тега Item в составе пакета UpdateObject, его задание указывает, что значение атрибута должно быть очищено.

ExistingOnly – используется при передаче тега Item в составе пакета UpdateObject, указывает, что значение атрибута изменяется только в том случае, если оно ранее уже было установлено.

AddValue - используется при передаче тега в составе пакета UpdateObject, указывает, что если значение или значения для атрибута уже были заданы, то новое значение не перезаписывает существующие, а добавляется к ним. Имеет смысл для многозначных атрибутов.

DelValue - при передаче тега в составе пакета UpdateObject – удалить указанное значение из уже заданных значений атрибута. Имеет смысл только для многозначных атрибутов.

Copy - при передаче тега в составе пакета UpdateObject. Позволяет скопировать значение одного атрибута (указанного в Copy) в значение другого (заданного в AttributeId). Копируются только те значения, которыми объект обладал перед началом выполнения запроса, значения, присваиваемые текущим запросом, не учитываются. По этой причине параметр имеет смысл только при редактировании, но не создании объекта.

Для мультиязычных свойств если для атрибута указана языковая версия (задан параметр Lang), то произойдет копирование только этой языковой версии; если Lang не задан, задан пустым или значение ALL, то произойдет копирование значений всех языковых версий. При этом принадлежность конкретного значения какой-либо языковой версии сохраняется такая же, как у исходного атрибута. Копирование между разными языковыми версиями одного и того же или разных атрибутов не предусмотрено.

Проверка совпадения типов атрибутов до начала копировании не производится, так как на практике в ряде случаев может потребоваться копирование несовпадающих по области значений атрибутов, присвоенные значения которых допускают автоматическое преобразование (например, числа или даты в строку и наоборот, если строковое значение фактически содержит в себе дату или число). В случае, если копируемое значение невозможно автоматически сконвертировать для присвоения новому атрибуту, будет возвращено сообщение об ошибке.

4.5. Запросы на изменение данных

Для всех запросов на редактирование обязательно задание кода информационной системы в атрибуте Originator корневого тега.

4.5.1. Редактирование/создание объекта

UpdateObject – запрос на создание/изменение конкретного объекта или набора объектов, заданных своими идентификаторами.

Параметры

Пакет содержит теги Item, описание которых приводится в структуре пакета Items, содержащего информацию о каком-либо объекте. В одном пакете может быть передан запрос на изменение сразу нескольких объектов. Для каждого объекта передается свой тег Item. Запросы на редактирования не могут выполняться от анонимного пользователя, задание идентификатора системы в атрибуте Originator для таких запросов обязательно.

Пример запроса на редактирование объекта в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test">
  <Item Code="Персона_1">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#comment" Value="Лучший сотрудник" />
  </Item>
</UpdateObject>
```

Пример запроса на редактирование объекта в формате JSON:

```
{"UpdateObject": {
 "Originator":"test",
"Item": [
    {
   "Code":"Персона_1",
     "Type": [
     {"TypeId":"Персона"}
   ],
   "Attribute": [
    {
      "Type":"Literal",
      "AttributeId":"http://www.w3.org/2000/01/rdf-schema#comment",
          "Value":"Лучший сотрудник"
          }
      ]
    }
  ]
 }
}
```

Пример запроса на создание объекта в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test">
  <Item LocalCode="1298">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="ФИО" Value="Хохолев Х.Х." />
  </Item>
</UpdateObject>
```

Перечень возможных атрибутов корневого тега:

Check - выполнить проверку правил логической целостности для всех объектов запроса;
Logic – применить правила получения логических выводов ко всем всех объектам запроса;

LogicRulesIds – идентификаторы (через запятую) применяемых правил логического вывода;
NotCheckMandatory – не учитывать обязательные атрибуты (для новых объектов);
HistoryDate - записать историю изменения указанной датой
ExecuteDate - отложить выполнение запроса до указанной даты
UsePlace – редактировать объекты только в хранилищах с указанным размещением;
CheckDuplicateOnly – выполнять изменения объектов, а только проверить переданные объекты на существование дубликатов (должны быть заданы логические правила для такой проверки);
Lang - языковая версия для всех атрибутов, переданных в запросе (редактирование объекта с учетом языковых версий описано в разделе 4.6.3).

Перечень атрибутов тега Item:

Code – идентификатор объекта;
LocalCode – идентификатор объекта в системе-источнике;
OperationId - уникальный идентификатор операции;
FullUpdate - полное обновление объекта, включая очистку атрибутов, отсутствующих в запросе;
CreateIfNotExists – создать, если не существует – флаг используется для создания новых объектов с предустановленным идентификатором;
Prefix – префикс генерации идентификатора нового объекта;
IgnoreTypes - не изменять принадлежность классам; список классов используется лишь для определения хранилища объекта;
AddTypes - вместо перезаписи принадлежности классам, добавить тип в список классов;
DomainIntersection - имеет смысл для свойств и свойств-ссылок: областью применения считать пересечение указанных классов, а не объединение (как по умолчанию);
RangeIntersection - имеет смысл для свойств и свойств-ссылок: областью значений свойства считать пересечение указанных классов, а не объединение;
Check - выполнить проверку логической целостности для конкретного объекта;
Logic – применить правила логических выводов для конкретного объекта;
LogicRulesIds – идентификаторы (через запятую) применяемых правил логического вывода;
HistoryDate - записать изменение указанной датой;
UsePlace – редактировать объект только в хранилищах с указанным размещением;
Lang - языковая версия для всех атрибутов объекта (описание см. в разделе 4.6.3).

Примечание ко всем: часть атрибутов (Check, Logic, Lang, HistoryDate и т.п.) могут быть заданы как в корневом теге, так и во вложенном теге Item. В случае, если атрибут указан в корневом теге, он относится ко всем объектам пакета. Но при этом если тот же атрибут задан далее в теге Item, то для данного объекта значение из тега Item перекрывает значение, заданное для всего пакета. Например, если в корневом теге передано Check=1, а для одного объекта указано Check=0, то проверка целостности будет выполнена для всех объектов кроме одного.

Более подробное описание атрибутов, относящихся только к тегу Item, приведено в описании тега Item в разделе 4.4.2. Далее приведена информация об атрибутах корневого тега UpdateObject, в том числе общих для него и тега Item.

Платформа АрхиГраф поддерживает два вида правил SHACL:

- правила контроля логической целостности (SHACL Constraints);
- правила получения логических выводов (SHACL Rules).

Для работы с правилами в модель должны быть импортированы специальные наборы классов. Настройка правил производится через интерфейс АрхиГраф.СУЗ.

В случае, если при редактировании объекта требуется применить логические правила того или иного вида к одному или нескольким объектам запроса, необходимо указывать следующие атрибуты, принимающие значения 0 и 1:

Check – чтобы применить правила контроля логической целостности;
Logic – применить правила получения логических выводов.

По умолчанию значения флагов Check и Logic равны 0, т.е. попыток поиска и применения логических правил не производится.

В случае, если требуется проверить не все, а лишь конкретные правила логического вывода, целочисленные идентификаторы этих правил указываются в строковом атрибуте **LogicRulesIds** в виде списка через запятую. Задать идентификаторы можно только для правил логического вывода, но не контроля целостности.

Если правило логической целостности нарушено, ответ будет дополнен атрибутом Message, содержащим сообщение нарушенного правила или правил. Изменение объекта при этом всё равно будет выполнено. Применение всех логических правил происходит после изменения объекта с учетом внесенных запросом правок.

Пример редактирования с применением логических правил:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test">
  <Item Code="ИвановИИ" Check="1">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="ДатаРождения" Value="1970-01-01"
/>
  </Item>
</UpdateObject>
```

Ответ:

```
<OperationResults Destination="test">
  <OperationResult Result="success" Code="ИвановИИ" Message="Для персоны не
задан ИНН"/>
</OperationResults>
```

В некоторых случаях при создании нового объекта нет возможности заполнить значения всех его атрибутов, в том числе могут быть не известны атрибуты, указанные как обязательные. В случае, если для объекта не заполнены обязательные атрибуты, будет возвращено сообщение об ошибке, и объект не будет создан. Для таких ситуаций можно воспользоваться флагом **NotCheckMandatory**, позволяющим отключить проверку

обязательных атрибутов при создании. Использование флага предполагает, что отсутствующие атрибуты будут заполнены при следующем редактировании объекта.

При создании, редактировании, удалении объекта в системе сохраняются метаданные об этом изменении, включающие в себя информацию о том, какой атрибут какого объекта был изменен, какой системой, в каком хранилище и так далее (работа с историей данных рассматривает в разделе 5.2). В качестве даты и времени изменения по умолчанию записываются текущие дата и время. Если необходимо зафиксировать дату и время изменения объекта в исходной системе, эти дату и время можно передать в запросе в атрибуте **HistoryDate** корневого тега или тега Item, в зависмости от того, относятся эти данные ко всем редактируемым объектам или только к конкретному.

Пример запроса с фиксированным временем изменения:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" HistoryDate="2020-01-01 12:00:00">
  <Item Code="ИвановИИ">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="РаботаетВ" Value="Альфа" />
  </Item>
</UpdateObject>
```

С помощью указания значения атрибута **ExecuteDate** можно запланировать запрос, который будет выполнен будущей датой. В случае поступления в систему запроса с заданным ExecuteDate, значение которого больше текущих даты и времени, запрос не выполняет, а помещается в специальную очередь. В качестве успешного результата выполнения операции будет возвращено значение *delayed* - отложен. В историю изменения объекта вносится запись о запланированном изменении. От записей, относящихся к фактическому изменению, записи в истории о будущих изменениях отличаются значением атрибута Plan: 0 для фактических и 1 для запланированных (запрос на получение истории изменения описан в разделе 5.2.1).

При наступлении запланированного момента запрос начинает выполняться. Дата фактического изменения так же будет записана в историю изменения объекта.
Пример запроса с отложенным временем выполнения:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" ExecuteDate="2030-01-01 00:00:00">
  <Item Code="ИвановИИ">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="РаботаетВ" Value="Бета" />
  </Item>
</UpdateObject>
```

Ответ:

```
<OperationResults Destination="test" >
  <OperationResult Result="delayed" Message="Saved successful" Code="
ИвановИИ "/>
</OperationResults>
```

Для хранилищ объектов может быть задано значение свойства Place – площадка. Запросы на редактирование объектов можно сузить до поиска и изменения объектов

только в хранилищах указанной площадки. Для этого имя площадки передается в атрибуте **UsePlace** корневого тега или тега Item.

Далее будут приведены разъяснения и примеры для различных атрибутов тегов Item вложенных в него.

Тег Item в рамках пакета на изменение может содержать значение атрибут **OperationId** – уникальный код операции. Строковый атрибут, генерируемый запрашивающей системой, не является обязательным. Но рекомендуется его указывать, поскольку данный идентификатор будет возвращен в ответе в теге OperationResult и позволяет сопоставить запрос на редактирование конкретного объекта с результатом выполнения этого запроса.

Пример запроса с OperationId:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" OperationId="000004">
  <Item LocalCode="1298" OperationId="0000041">
    <Type TypeId="Персона" />
    <Attribute Type="Literal" AttributeId="ФИО" Value="Иванов" />
  </Item>
<Item LocalCode="1299" OperationId="0000042">
    <Type TypeId="Персона" />
  </Item>
</UpdateObject>
```

Ответ:

```
<OperationResults Endpoint="demo" Destination="test" OperationId="55555" >
  <OperationResult Result="success" Code="Персона_1" OperationId="0000041"
LocalCode="1298"/>
  <OperationResult Result="error" Message="Bad values count 0 for property
&amp;#39;ФИО&amp;#39;" ErrorCode="267" OperationId="0000042" LocalCode="1299"/>
</OperationResults>
```

При редактировании объектов по умолчанию будут изменены только те атрибуты, которые были переданы в тегах Attribute. Отсутствующие в запросе атрибуты объекта сохранят свои значения. Если требуется оставить у объекта только переданные атрибуты и удалить все прочие, то необходимо указать флаг **FullUpdate**=1 в теге Item.

Для очистки значений конкретного атрибута используется флаг Empty в теге Attribute. Пример запроса, при котором происходит удаление всех значений атрибута Адрес:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Организация_1"   OperationId="0000041">
    <Type TypeId="Организация" />
    <Attribute Type="Literal" AttributeId="Адрес" Empty="1" />
  </Item>
</UpdateObject>
```

Если атрибут принимает множественные значения, то при редактировании будут удалены все старые значения атрибута и присвоены значения, переданные в запросе. Для добавления значений к списку уже заданных ранее значений необходимо использовать флаг AddValue для тега Attribute.

Пример использования AddValue. Пусть у объекта *Организация_1* значение атрибута УчаствуетВПроекте равно *Проект_1.* В результате выполнения запроса

```
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Организация_1"   OperationId="0000041">
    <Type TypeId="Организация" />
    <Attribute Type="Reference" AttributeId="УчаствуетВПроекте"
Value="Проект_2"/>
  </Item>
</UpdateObject>
```

атрибут УчаствуетВПроекте примет значение *Проект_2.* При выполнении же запроса

```
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Организация_1"   OperationId="0000041">
    <Type TypeId="Организация" />
    <Attribute Type="Reference" AttributeId="УчаствуетВПроекте"
Value="Проект_2" AddValue="1" />
  </Item>
</UpdateObject>
```

атрибуту УчаствуетВПроекте будет иметь два значения *Проект_1* и *Проект_2.*

Флаг Ignore тега Attribute означает, что данный атрибут не нужно учитывать при редактировании; теги Attribute с данным флагом будут пропущены при обработке пакета.

При редактировании объекта может быть использован флаг IgnoreTypes для тега Item. В этом случае вложенные теги Type можно передавать или не передавать – в любом случае объект сохранит принадлежность тем классам, которая у него была. Если же флаг IgnoreTypes не указан, то теги Type являются обязательными и произойдет присвоение объекту списка классов, переданных в запросе. При создании нового объекта атрибут Type является обязательным, а использование флага IgnoreTypes приведет к возвращению ошибки.

Другой флаг, влияющий на обработку принадлежности объекта классам - AddTypes. При задании AddTypes=1 для существующего объекта список классов, которым он принадлежит, будет дополнен списком переданных в запросе классов. Для нового объекта произойдет присвоение списка переданных классов так же, как без флага AddTypes. При одновременном задании значения 1 для флагов IgnoreTypes и AddTypes флаг AddTypes будет проигнорирован.

Ответ

Пакет **OperationResults** - пакет с информацией о результатах выполнения операции.

Вложенные теги: OperationResult – результат выполнения запроса для отдельного объекта.

Атрибуты тега OperationResult:

OperationId – идентификатор запроса, значение из тега Item входного пакета.

Result – может вернуть значения *success*, *error* либо *proposed*. Значение *success* вернется в случае успешного изменения объекта, значение *error* - в случае какого-либо сбоя, возникшего при выполнении запроса. В случае, если система, выполняющая запрос, имеет к изменяемому объекту права на редактирование с подтверждением, и

произошло успешное добавление запроса на подтверждение в базу, в атрибуте Result будет возвращено значение *proposed*.

Message – текст сообщения об ошибке в случае сбоя либо поясняющий операцию текст в других случаях; например, при групповой операции Message будет содержать информацию о числе измененных или удаленных объектов.

Code – код АрхиГраф измененного/созданного объекта

LocalCode - код в системе источники объекта - в случае, если данное значение было передано в запросе.

Пример ответа

В формате XML

```
<OperationResults Destination="test" >
  <OperationResult Result="success" OperationId="0000041" Code="Персона_1"
LocalCode="1297" />
  <OperationResult Result="error" Message="Unknown code Персона_007"
OperationId="0000042" Code="Персона_007" />
</OperationResults>
```

Пример ответа в формате JSON:

```
{"OperationResults":  {
 "Originator":"test",
 "OperationId":"000004",
 "OperationResult":   [
    {
    "Result":"success",
    "Message":"Unknown code Персона_007",
      "OperationId":"0000041",
    "Code":"Персона_1",
    "LocalCode":"1297"
  },
    {
    "Result":"error",
    "Message":"Unknown code Персона_007",
      "OperationId":"0000042",
    "Code":"Персона_007"
  }
  ]
 }
}
```

Кроме того, после успешного выполнения операции АрхиГраф отправляет пакет SubscriptionItems, который получают все информационные системы, заинтересованные в сведениях об объектах такого типа. Формат пакета SubscriptionItems полностью повторяет формат пакета Items, описанного в предыдущем разделе. Для объектов в пакете SubscriptionItems передается полный набор их атрибутов, а не только измененные.

4.5.2.Удаление объекта

DeleteObject - запрос на удаление конкретного объекта, заданного идентификатором.

Параметры

Code – обязательный атрибут - код АрхиГраф удаляемого объекта.

Ответ

Пакет **OperationResults** - пакет с информацией о результатах выполнения операции. Кроме того, после успешного выполнения операции АрхиГраф отправляет информационным системам, подписанным на изменение объектов данного типа, пакет SubscriptionDeleteItems. Структура пакета идентична пакету Items, он содержит тег Item с описанием удаляемой сущности.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeleteObject Code="ИвановИИ" Originator="test" />
```

В формате json:

```
{"DeleteObject":{"Code":"ИвановИИ", "Originator":"test" }}
```

Дополнительные необязательные атрибуты:

CheckAllStorages – флаг, принимает значения 0 и 1: искать объект во всех хранилищах, к том числе тех, в которых по текущему состоянию информационной модели он не должен находиться. Позволяет исправить ошибки в физическом размещении объектов.
ExecuteDate – атрибут типа дата и время: поместить запрос в очередь на выполнение будущей датой.
HistoryDate – тип дата и время: записать в историю изменение указанной датой.
DeleteReference – флаг со значениями 0 и 1: после удаления объекта удалить все ссылки на него. По умолчанию при удалении объекта не происходит автоматического удаления значений свойств-ссылок, указывающих на объект. В случае, если в запросе DeleteObject передан флаг DeleteReference=1 и удаление объекта прошло успешно, во внутреннюю очередь будет поставлена задача на удаление всех ссылок на данный объект.
VerifyReference – флаг, значения 0 и 1: проверить наличие ссылок на объект. В случае, если будет найден хотя бы один объект, значение атрибута которого равно удаляемому объекту, удаление объекта не будет проведено. В сообщение об ошибке будет указан идентификатор первого найденного ссылающегося объекта.

Пример для флага VerifyReference:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeleteObject Code="Альфа" Originator="test" VerifyReference="1" />
```

Ответ:

```
<OperationResults Destination="test" >
  <OperationResult Result="error" Message="Object ИвановИИ refers to Альфа"
  ErrorCode="230" Code="Альфа"/>
</OperationResults>
```

UsePlace – строковый: название площадки, хранилищами которой следует ограничить поиск и удаление объектов.

4.5.3. Редактирование набора объектов

UpdateObjectsGroup – запрос на изменение группы объектов.

Параметры

Запрос имеет два формата — полный и сокращенный, аналогичные форматам запроса GetObjectsGroup.

В сокращенном формате запрос состоит из тега UpdateObjectsGroup и одного или нескольких вложенных в него тегов Attribute, характеризующих те атрибуты объектов, которые будут изменены. Описание тега Attribute приводится в структуре пакета Items. Для тега UpdateObjectsGroup задается дополнительный атрибут Code – идентификатор класса модели. При выполнении запроса краткого формата будут изменены все объекты, относящиеся к указанному классу.

В полном формате в отличие от краткого тег UpdateObjectsGroup не имеет атрибута Code, а вместо этого включает в себя несколько вложенных тегов ObjectType, FilterGroup, Item, Geometry, позволяющих задать условие для отбора тех объектов, которые будут изменены. Синтаксис тегов ObjectType, FilterGroup, Item, Geometry приведен в структуре пакета GetObjectsGroup.

Помимо этого, запрос может содержать атрибуты Check, Logic, LogicRulesIds, ExecuteDate, HistoryDate, UsePlace и IgnoreTypes, аналогичные соответствующим атрибутам запроса UpdateObject.
Специфичным для запроса UpdateObjectsGroup является флаг **AddHistory**, используемый в сочетании с атрибутом ExecuteDate (выполнение запроса будущей датой). В отличие от запроса UpdateObject, при постановке задачи на редактирование набора объектов будущей датой по умолчанию не записывается плановая история изменений. Если планируемое изменение требуется зафиксировать для каждого объекта, удовлетворяющего в текущий момент условиям отбора, необходимо в запрос дополнительно передать AddHistory=1. Следует учитывать, что набор объектов, удовлетворяющих условиям отбора в момент внесения запроса и в момент его фактического выполнения могут отличаться. Изменения будут выполнены только на тех объектах, которые будут удовлетворять условиям отбора на момент фактического выполнения запроса на групповое изменение.

Ответ

Пакет **OperationResults** – пакет с информацией о результате выполнения операции. Его атрибуты:
Count – количество объектов, удовлетворяющих условиям фильтрации запроса;
UpdateCount – количество успешно измененных объектов;
Message – текстовое сообщение с информации о количестве успешно измененных объектов.

В случае, если найдены удовлетворяющие условиям отбора объекты, информация об обработке каждого из них будет возвращена в теге *OperationResult*.

Также АрхиГраф рассылает пакеты SubscriptionItems со всеми измененными в результате выполнения запроса объектами. Пакеты SubscriptionItems получат все информационные системы, имеющие право доступа к этим объектам.

Примеры запросов

Краткий формат:

```
<?xml version="1.0" encoding="UTF-8"?>
 <UpdateObjectsGroup Code="Компании" Endpoint="demo" Originator="test"
OperationId="0945ab454">
 <Attribute Type="Literal" AttributeId="ДействующаяКомпания" Value="true" />
</UpdateObjectsGroup>
```

Полный формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObjectsGroup Endpoint="demo" Originator="test"
OperationId="0945ab4fr4">
  <ObjectType Code="Компании"/>
  <ObjectType Code="Организации"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="Название" Value="ООО Альфа" Comparison="Equal" />
    <Filter Attribute="Название" Value="ООО Бета" Comparison="Equal" />
  </FilterGroup>
  <Attribute Type="Literal" AttributeId="ДействующаяКомпания" Value="false"
/>
</UpdateObjectsGroup>
```

Пример ответа

```
<OperationResults Count="3" UpdateCount="2" Message="Изменено 2 объекта"
Endpoint="demo" Destination="test" >
  <OperationResult Result="success" Code="Компания_00001" />
  <OperationResult Result="error" Code="Компания_00002" Message="Произошел
сбой при сохранении изменений" />
  <OperationResult Result="success" Code="Компания_00003" />
</OperationResults>
```

4.5.4. Удаление набора объектов

DeleteObjectsGroup – запрос на удаление группы объектов.

Параметры

Запрос имеет два формата — полный и сокращенный, аналогичные форматам запроса GetObjectsGroup.

В сокращенном формате запрос состоит из тега DeleteObjectsGroup, имеющего дополнительный атрибут Code – идентификатор класса объектов. При выполнении запроса краткого формата будут удалены все объекты, относящиеся к указанному классу.

В полном формате тег DeleteObjectsGroup включает в себя несколько вложенных тегов ObjectType, FilterGroup, Item, Geometry, позволяющих задать условие для отбора тех объектов, которые будут изменены. Синтаксис тегов ObjectType, FilterGroup, Item, Geometry приведен в структуре пакета GetObjectsGroup.

Также запрос может содержать атрибуты ExecuteDate и HistoryDate аналогичные соответствующим атрибутам запроса UpdateObject. Как и для запроса UpdateObjectsGroup, при выполнении запроса будущей датой по умолчанию плановая история изменения не записывается; для ее фиксации требуется использовать флаг AddHistory. Флаги DeleteReference и VerifyReference работают аналогично одноименным атрибутам запроса DeleteObject.

Ответ

Пакет **OperationResults** – информация о результате выполнения операции. Его атрибуты:

Count – количество объектов, удовлетворяющих условиям фильтрации запроса;
DeleteCount – количество успешно удаленных объектов;
Message – текстовое сообщение с информации о количестве удаленных объектов.

В случае, если найдены удовлетворяющие условиям отбора объекты, информация об обработке каждого из них будет возвращена в теге *OperationResult*.

Также платформа рассылает пакеты SubscriptionDeleteItems со всеми удаленными в результате выполнения запроса объектами. Пакеты SubscriptionDeleteItems получат все информационные системы, заинтересованные в сведениях об объектах, удовлетворяющим условиям запроса.

Примеры запросов

Краткий формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeleteObjectsGroup Code="Компании" Endpoint="demo" Originator="test"
OperationId="0945ab454dc">
</DeleteObjectsGroup>
```

Полный формат:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeleteObjectsGroup Endpoint="demo" Originator="test"
OperationId="0945ab454dcf7" CombineGroups="Or">
  <ObjectType Code="Компании"/>
  <ObjectType Code="Организации"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="Название" Value="ООО Альфа" Comparison="Equal"/>
    <Filter Attribute="Название" Value="ООО Бета" Comparison="Equal"/>
  </FilterGroup>
</DeleteObjectsGroup>
```

Пример ответа

```
<OperationResults Count="2" DeleteCount="2" Message="Удалено 2 объекта"
Endpoint="demo" Destination="test" >
  <OperationResult Result="success" Code="Компания_00001" />
  <OperationResult Result="success" Code="Компания_00002" />
</OperationResults>
```

4.5.5. Объединение объектов-дубликатов

MergeObject – запрос на объединение пар объектов-дубликатов.

Запрос служит для слияния значений двух объектов, которые пользователь счет дублирующими. В зависимости от параметров запроса, помимо дополнения основного объекта значениями атрибутов объекта-дубликата, может происходить так же удаление дублирующего объекта.

Параметры

Пакет содержит теги *Item*, каждый из которых передает информацию о паре объединяемых объектов. В одном пакете может быть передан запрос на изменение сразу нескольких пар дубликатов, каждая передается в своем теге Item. Задание идентификатора системы в атрибуте Originator для таких запросов обязательно.

Параметры тега Item:

Code – код основного объекта, в который буду вливаться значения атрибутов дубликата, обязательный атрибут.
Duplicate – код объекта-дубликата, обязательный атрибут.
DeleteDuplicate – флаг 0/1, удалять ли объект-дубликат в результате выполнения операции. По умолчанию значение равно 0, дубликат не удаляется.
DeleteDuplicateReference – флаг 0/1, удалять ли у объекта-дубликата значение атрибута archigraph:duplicate, указывающее на основную запись.

MergeTypes – флаг, объединять ли принадлежности множествам классов модели, к которым принадлежат объект и его дубликат. По умолчанию значение равно 0, и объединяемый объект сохраняет свои принадлежности классам.

MergeEmptyOnly – флаг, присваивать ли значения только тех атрибутов, которые не заданы у основного объекта. По умолчанию принимает значение 0. Если флаг не указан или передано значение 0, и у объектов установлены неравные значения одного и того же однозначного атрибута, то будет возвращено сообщение об ошибке. Исключением являются атрибуты archigraph:SourceSystem (система-источник) и archigraph:LocalCode (код в системе-источнике), которые пропускаются в любом случае.

MergeMultivalue – флаг, для атрибутов, принимающих множественные значение, объединять значения атрибута основного объекта со значениями того же атрибута объекта-дубликата. Если заданы одновременно флаги MergeEmptyOnly и MergeMultivalue, то заданные у основного объекта атрибуты, принимающие единственное значение, будут пропущены, а значения атрибутов, принимающие несколько значений, будут объединены.

FullUpdate – флаг, выполнять ли полное обновление основного объекта. По умолчанию значение 0, если передано 1, то у объединяемого объекта будут очищены все ранее заданные атрибуты, включая те, которые не переданы запросом.

AllAttributes – флаг, присваивать значения всех атрибутов, заданных у объекта-дубликата. По умолчанию значение равно 0, при этом присваиваются только атрибуты, переданные в тегах Attribute.

Тег *Type* с атрибутом **TypeId** описывает те классы модели, которые присущи объекту-дубликату и которые необходимо присвоить объединяемому с ним объекту. Учитывается только при передаче значения флага MergeTypes равным 1.

Теги *Attribute* содержат описание тех атрибутов объекта-дубликата, значения которых необходимо изменить у основного объекта. Описание тега Attribute аналогично одноименному тегу пакета UpdateObject. Если в теге Attribute не заданы атрибуты Value и Empty, а лишь идентификатор свойства в атрибуте AttributeId, то значение для присваивания будет взято из значений объекта-дубликата. В случае, если в теге Item передано значение флага AllAttributes равное 1, теги Attribute будут проигнорированы.

Примеры запросов

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<MergeObject Endpoint="demo" Originator="test">
   <Item  Code="Персона_001" Duplicate="Персона_002" AllAttributes="1"
MergeEmptyOnly="1" DeleteDuplicate="1" />
</MergeObject>
```

В формате json:

```
{"MergeObject":{ "Endpoint":"demo", "Originator":"test", "Item": [{
"Code":"Персона_001", "Duplicate": "Персона_002", "AllAttributes": 1,
"MergeEmptyOnly":1, "DeleteDuplicate": 1 }]}}
```

В формате XML с указанием атрибутов, которые необходимо слить:

```
<MergeObject Endpoint="demo" Originator="test">
 <Item Code="Контрагент_001" Duplicate="ЮридическоеЛицо_001" MergeTypes="1"
MergeEmptyOnly="1" MergeMultivalue="1">
    <Type TypeId="ЮридическоеЛицо" />
    <Attribute AttributeId="инн" />
    <Attribute AttributeId="огрн" />
    <Attribute AttributeId="телефон" />
  </Item>
</MergeObject>
```

Если задана подписка информационных систем на изменение классов, которым принадлежат объединяемые объекты, то пакеты, отправляемые по подписке, будут дополнены специальными атрибутами:
- пакет об изменении или удалении дубликата дополняется атрибутом MergeTo, в котором указан идентификатор основного объекта;
- пакет об изменении основного объекта содержит идентификатор объекта-дубликата в атрибуте MergeFrom.

4.6. Языковые версии данных

4.6.1. Получение списка поддерживаемых языков

GetLanguages – получить перечень языковых версий, поддерживаемых системой. Метод не имеет собственных параметров.

Примеры запросов

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetLanguages Endpoint="demo"  Originator="test" />
```

В формате json:

```
{"GetLanguages":{ "Endpoint":"demo", "Originator":"test"}}
```

Ответ

Пакет **LanguagesList** – список поддерживаемых языков.
Вложенные теги *Language* характеризуют конкретную языковую версию и имеет следующие атрибуты:
Code – код языковой версии в стандарте ISO 639.

Name – читаемое наименование языковой версии.

Default – является ли указанный язык языком по умолчанию, возвращаемое значение false или true.

Пример ответа

В формате XML:

```
<LanguagesList Endpoint="demo" Destination="system">
  <Language Code="RU" Name="Русский" Default="true" />
  <Language Code="EN" Name="English" Default="false" />
</LanguagesList>
```

В формате json:

```
{
  "LanguagesList":{
    "Endpoint": "demo", "Destination": "system",
    "Language": [
      {"Code": "RU", "Name": "Русский", "Default": "true"}
      {"Code": "EN", "Name": "English", "Default": "false"}
    ]
  }
}
```

4.6.2. Получение объектов с учетом языковых версий

Настройками MDM один из языков задан как язык по умолчанию. При запросе объектов если языковая версия не указана явно и для объекта есть данные на нескольких языках, то будут возвращены только данные версии языка по умолчанию.

Для запроса данных на других языках служит атрибут **Lang**, указываемый в корневом запросе. В качестве значения атрибута может быть указан код одного из языков, поддерживаемых системой, либо ключевое слово ALL, позволяющее получить по объекту все заданные для него языковые версии.

Примеры запросов

Получить данные по объекту на английском языке, запрос в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObject Endpoint="demo" Originator="system" Code="Персона" Lang="EN" />
```

В формате json:

```
{
  "GetObject":{
   "Endpoint":"demo",
    "Originator": "system",
    "Code": "Персона",
    "Lang": "EN"
  }
}
```

Получить полные данные по объекту, запрос в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObject Endpoint="demo" Originator="system" Code="Персона" Lang="ALL" />
```

Для объектов указанного класса получить значения атрибутов на английском языке, запрос в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup Endpoint="demo" Originator="system" Code="Персона" Lang="EN" />
```

В случае, если свойство объекта имеет значения на разных языках, языковая версия указывается в атрибуте Lang тега Attribute.

Пример ответа в формате XML, содержащего описание объекта, для свойства rdfs:label которого определены значения на двух языках:

```
<Items Endpoint="demo" Destination="system" Lang="ALL" >
  <Item Code="Персона" Name="Персона">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Класс"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Персона" Lang="RU"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Person" Lang="EN"/>
  </Item>
</Items>
```

Если язык данных совпадает с языком по умолчанию, атрибут Lang для него может не указываться. Пример ответа в случае, когда русский язык является языком по умолчанию:

```
<Items Endpoint="demo" Destination="system" Lang="ALL" >
  <Item Code="Персона" Name="Персона">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Класс"/>
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Персона" />
    <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Person" Lang="EN"/>
  </Item>
</Items>
```

Языковые версии применимы только к атрибутам-литералам строкового типа. Атрибуты других типов будут возвращены при любом запросе объекта. Для атрибутов-ссылок

читаемое наименование ссылки будет возвращено на языке по умолчанию, если языковая версия в запросе не указана либо выбран вариант Lang=ALL. Если запрошены данные на конкретном языке и значение rdfs:label на этом языке для значения ссылки задано, в атрибуте Name тега Item будет возвращено оно. Если читаемого наименования на запрошенном языке у значения ссылки нет, то в атрибуте Name тега Item будет возвращено наименование на языке по умолчанию.

Пример описания объекта, полученного при запросе без указания атрибута Lang, язык по умолчанию – русский:

```
<Item Code="Персона_1" Name="Тимофеев Т.Т.">
  <Type TypeId="Персона" Name="Персона"/>
  <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Тимофеев Т.Т."/>
  <Attribute AttributeId="ДатаРождения" Type="Literal" Value="2001-01-02"/>
  <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Компания_1" Name="
Иванов и партнеры"/>
</Item>
```

Пример описания объекта, полученного при запросе с заданием Lang=EN:

```
<Item Code="Персона_1" Name="Timofeev T.">
  <Type TypeId="Персона" Name="Person"/>
  <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Timofeev T." Lang="EN"/>
  <Attribute AttributeId="ДатаРождения" Type="Literal" Value="2001-01-02"/>
  <Attribute AttributeId="РаботаетВ" Type="Reference" Value="Компания_1"
Name="Ivanoff Company Inc"/>
</Item>
```

4.6.3. Редактирование языковых версий данных

При редактировании объектов язык, к которому относится указанное значение свойства, указывается в атрибуте Lang тега Attribute. Так же языковая версия может быть указана в атрибуте Lang тега Item, в этом случае она распространяется на все переданные в запросе свойства.

Примеры запросов

Присвоить для объекта читаемое наименование нескольких языках, на запрос в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Endpoint="demo" Originator="system"
  <Item Code="Персона" OperationId="1234">
    <Type TypeId="Проект" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Value="Персона" Lang="RU" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Value="Person" Lang="EN" />
 </Item>
</UpdateObject>
```

Присвоить для объекта значения нескольких атрибутов на английском языке, на запрос в формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Endpoint="demo" Originator="system"
```

```
  <Item Code="Персона" OperationId="1234" Lang="EN">
    <Type TypeId="Проект" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Value="Person" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#comment" Value="Class comment" />
 </Item>
</UpdateObject>
```

Языковые версии применимы только к атрибутам-литералам строкового типа. Если язык данных совпадает с языком по умолчанию, то его можно не указывать. Если языковая версия указана для свойства, тип которого не допускает данные на разных языках (числовой, дата и т.п.), то будет возвращена ошибка.

Пример ошибки, возвращаемой в случае попытки присвоить языковую версию свойству типа boolean:

Запрос:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Endpoint="demo" Originator="test" OperationId="000004">
  <Item  Code="Персона" OperationId="0000041">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" />
      <Attribute AttributeId="http://trinidata.ru/archigraph-mdm/archive"
Type="Literal" Value="false" Lang="EN"/>
  </Item>
</UpdateObject>
```

Ответ:

```
<OperationResults Endpoint="demo" Destination="test" OperationId="000004" >
  <OperationResult Result="error" Message="Language versions not allowed for
attribute &#39;http://trinidata.ru/archigraph-mdm/archive&#39;"
OperationId="0000041" Code="Персона"/>
</OperationResults>
```

4.7. Запросы для работы с подписками

4.7.1. Установить подписку

UpdateSubscription – подписать информационную систему на получение информации об изменении объектов заданного класса и/или на изменение модели самого класса.

Параметры

Вложенный тег: *Subscribe*

Атрибуты тега Subscribe:

Format – формат получаемых пакетов с информацией об объектах, принимает значение XML либо JSON. Является обязательным атрибутом при создании новой подписки информационной системы на заданный класс. Может не указываться при изменении параметров уже существующей подписки.

OperationId – необязательная строка. Значение, которое будет передаваться в атрибуте OperationId пакетов SubscriptionItems и SubscriptionDeleteItems. Служит для различения пакетов, полученных по разным подпискам.

Delayed – необязательный атрибут, принимает значения 0 (используется по умолчанию) либо 1. Является ли рассылка отложенной. Если значение равно 0, то отправка пакета с измененными объектами будет производиться сразу после внесения изменений. В случае, если нет необходимости в немедленном отклике, можно использовать отложенную отправку, что снижает нагрузку на систему и не приводит к замедлению скорости обработки запросов на стороне АрхиГраф.

Objects – необязательный атрибут, принимает значения 0 (используется по умолчанию) или 1. Получать ли информацию об изменении индивидуальных объектов, принадлежащих классам, указанным в подписке

Model – необязательный атрибут, принимает значения 0 (по умолчанию) или 1. Получать ли информацию об изменении свойств самого класса или его атрибутов.

Active – необязательный, по умолчанию равен 1 – подписка является активной (значение 1) или не активной (значение 0).

Exclude – необязательный параметр, принимает значения 0 и 1, по умолчанию равен 0. Позволяет исключить подкласс из подписки. Функционал отправки по подписке учитывает наследование классов: если Класс2 является подклассом Класс1, и система подписалась на получение изменения объектов Класс1, то система получит и изменения объектов Класс2. В случае, если получение объектов Класс2 не требуется, необходимо дополнительно задать подписку на Класс2 с указанием Exclude=1.

Host, **Port**, **Login**, **Password**, **Queue** – реквизиты очереди для получения пакетов об изменении сущностей. Если ни один из данных атрибутов не передан, но для системы уже есть зарегистрирована активная подписка, то будут использованы реквизиты существующей подписки.

Broker – необязательный атрибут, брокер сообщений. Поддерживаемые значения RabbitMQ и ApacheKafka. Если не указан, то будет использовано значение, заданное настройками MDM как значение брокера по умолчанию.

Originator, Token – необязательные строковые атрибуты. Если заданы, то будут переданы в корневом теге пакета, получаемого по подписке.

Тег *ObjectType* – вложен в тег Subscribe.

Перечисляет классы модели, на изменение объектов или свойств которых производится подписка.

Атрибуты тега ObjectType:

Code – идентификатор класса

Пример запроса

Запрос на добавление к существующим подпискам подписки на получение информации об объектах классов Контрагент и Сотрудник:

```
<UpdateSubscription Endpoint="demo" Originator="test">
 <Subscribe Format="json" Delayed="0" Objects="1" >
  <ObjectType Code="Контрагент" />
  <ObjectType Code="Сотрудник" />
```

```
</Subscribe>
</UpdateSubscription>
```

Ответ

Пакет **OperationResults** – информация о результате выполнения операции.

Чтобы подписаться на получение информации об изменении всех классов модели, в качестве класса в теге ObjectType указывается служебное слово __root__.

Пример запроса

Запрос на получение информации обо всех изменениях модели:

```
<UpdateSubscription Endpoint="demo" Originator="system">
 <Subscribe Format="json" Delayed="0" Objects="0" Model="1" Host="127.0.0.1"
Port="5672" Login="test" Password="test" Queue="test_subscribe">
  <ObjectType Code="__root__" />
</Subscribe>
</UpdateSubscription>
```

4.7.2. Получить статус подписки

GetSubscription – для заданных классов получить информацию о том, подписана ли система на получение информации об изменении сущностей и свойства этой подписки. Если классы не заданы, будет возвращена информация обо всех подписках системы.

Параметры

Тег *ObjectType*

Атрибуты тега ObjectType:

Code – идентификатор класса

Пример запроса

В формате XML:

```
<GetSubscription Endpoint="demo" Originator="test">
  <ObjectType Code="Сотрудник" />
</GetSubscription>
```

В формате JSON:

```
{"GetSubscription":{ "Endpoint":"demo", "Originator":"test",
"ObjectType":[{"Code":"Сотрудник"}]}}
```

Ответ

Пакет **Subscribes**

Вложенные теги

Subscribe – характеризуют отдельную подписку. Если для системы нет зарегистрированных подписок для указанного в запросе класса, то тег Subscribes не будет содержать вложенных тегов.

Атрибуты тега Subscribe:

Active – подписка является активной (значение 1) или не активной (значение 0);

Format – формат получаемых пакетов, XML либо JSON;

OperationId – значение для атрибута OperationId пакетов SubscriptionItems и SubscriptionDeleteItems;

Delayed – является ли рассылка отложенной;

Objects - получать ли информацию об изменении индивидуальных объектов;

Model – получать ли информацию об изменении модели класса;

Host, **Port**, **Login**, **Password**, **Queue**, **Broker** – реквизиты очереди для получения пакетов об изменении.

Вложенные теги: *ObjectType*

Атрибуты тега ObjectType:

Code – идентификатор класса
Name – читаемое наименование класса

Пример ответа

В формате XML:

```
<Subscribes Endpoint="demo" Destination="test">
  <Subscribe Format="json" OperationId="" Delayed="0" Objects="1" Model="0"
Host="12.12.12.123" Port="15672" Queue="MDM_IN" >
    <ObjectType Code="Контрагент" Name="Контрагенты" />
    <ObjectType Code="Сотрудник" Name="Сотрудники" />
  </Subscribe>
</Subscribes>
```

Подписки учитывают иерархию классов. Если Класс2 является подклассом для Класс1, то к объектам и свойствам класса Класс2 будут применены все подписки, оформленные для класса Класс1. При запросе состояния подписок класса Класс2 будет возвращено описание фактической подписки.

Пример запроса для подкласса:

```
<GetSubscription Endpoint="demo" Originator="test">
  <ObjectType Code="Класс2" />
</GetSubscription>
```

Ответ при запросе подписки подкласса:

```
<Subscribes Endpoint="demo" Destination="test">
  <Subscribe Format="json" [ … ]>
    <ObjectType Code="Класс1" Name="Класс 1" />
  </Subscribe>
</Subscribes>
```

4.7.3. Отменить подписку

DeleteSubscription – отменить подписку системы на получение информации по указанным классам.

Вложенные теги: *ObjectType*

Атрибуты ObjectType:

Code – обязательный, идентификатор класса

Не обязательные атрибуты:

Format – позволяет удалить только подписки, возвращающие данные в указанном формате JSON или XML. Если не задан, удаляются подписки всех форматов.

Delayed – позволяет удалить только немедленные (если передано значение 0) или отложенные (передано значение 1) подписки. Если не передан, удаляются подписки всех типов.

Objects – если передано значение 1, то будут исключены подписки на получение изменений объектов указанного класса.

Model – если передано значение 1, то будут удалены подписки на получение изменений модели

Если не передано ни Objects=1, ни Model=1, то будут удалены подписки как на получение изменений объектов, так и изменений модели.

Пример запроса

В формате XML:

```
<DeleteSubscription Endpoint="demo" Originator="test">
  <ObjectType Code="Сотрудник" />
</DeleteSubscription>
```

Ответ

Пакет **OperationResults** – информация о результате выполнения операции.

5. API работы с историйностью данных и модели

5.1. Работа с историей модели

5.1.1. Создание виртуальной точки доступа

CreateEndpoint – создать виртуальную точку доступа с состоянием модели на определенную дату.

Параметры

Date – момент времени в прошлом, состояние модели на который необходимо вернуть

Пример запроса

В формате XML:

```
<CreateEndpoint Endpoint="demo" Originator="test" Date="2019-01-01 00:00:00"
/>
```

Пример запроса состояния модели на 01.01.2019:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetDataSchema Endpoint="demo_20190101000000_24234sad34" Destination="test"
/>
```

Ответ

Пакет **OperationResults** – информация о результате выполнения операции. В случае успеха в атрибуте Code будет возвращен идентификатор созданной точки доступа. Этот идентификатор в дальнейшем можно использовать в качестве значения атрибута Endpoint в запросах на получение модели данных (GetDataSchema, GetDataSchemaCompact).

Пример ответа

```
<OperationResults Endpoint="demo" Destination="test" >
  <OperationResult Result="success" Code="demo_20190101000000_24234sad34" />
</OperationResults>
```

5.1.2. Удаление виртуальной точки доступа

DeleteEndpoint – удалить виртуальную точку доступа.

Параметры

Code – код виртуальной точки доступа

Пример запроса

В формате XML:

```
<DeleteEndpoint Originator="test" Code="demo_20190101000000_24234sad34" />
```

Ответ

Пакет **OperationResults** – информация о результате выполнения операции.

Пример ответа

В формате XML:

```
<OperationResults Destination="test" >
  <OperationResult Result="error" Message="Точка доступа
demo_20190101000000_24234sad34 не найдена"
Code="demo_20190101000000_24234sad34" />
</OperationResults>
```

5.2. Работа с историей данных

5.2.1. Получить историю изменений элемента данных

GetObjectHistory – запрос на получение истории изменения объекта

Параметры

Code – код объекта, информацию о котором необходимо получить.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectHistory Originator="test" Code="ИвановИИ" />
```

В формате JSON:

{"GetObjectHistory": {"Originator":"test","Code":"ИвановИИ"}}

Ответ

Пакет **Items** – описание объекта.

Пример ответа

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И. И.">
   <Type>
    <Operation Date="2019-01-02T16:20:00" System="test"
OperationId="918273645" Plan="0"/>
     <Set Value="Персона"/>
     <Set Value="Сотрудник"/>
    </Operation>
    <Operation Date="2019-01-01T00:00:00" System="test" Plan="0">
     <Set Value="Персона"/>
    </Operation>
   </Type>
   <Attribute Type="Literal" AttributeId="ФИО">
    <Operation Date="2019-01-01T00:00:00" System="test" Plan="0">
     <Set Value=" Иванов И.И."/>
    </Operation>
   </Attribute>
   <Attribute Type="Literal" AttributeId="ДатаРождения">
    <Operation Date="2019-01-01T00:00:00" System="test" Plan="0">
     <Set Value="1979-01-01"/>
    </Operation>
   </Attribute>
   <Attribute Type="Reference" AttributeId="РаботаетВ">
    <Operation Date="2019-01-03T00:00:00" System="test" Plan="0">
     <Set Value="ОООБета"/>
    </Operation>
    <Operation Date="2019-01-02T16:20:00" System="test"
OperationId="918273645" Plan="0">
     <Set Value="ОООАльфа"/>
    </Operation>
   </Attribute>
  </Item>
</Items>
```

Пример ответа в формате JSON:

```
{"Items": {
  "Destination":"test",
  "Item": [
```

```
{
"Code": "ИвановИИ",
"Name": "Иванов И.И.",
"Type": [
      {
          "Operation":[
             {
               "Date":"2019-01-02 00:00:00",
               "System":"test",
               "Operationid":"918273645",
               "Plan":0,
               "Set":[{"Value":"Персона"},{"Value":"Сотрудник"}]
             },
             {
               "Date":"2019-01-01 00:00:00",
               "System":"test",
               "Set":[{"Value":"Персона"}]
             }
           ]
      }
],
"Attribute": [
     {
        "Type":"Literal",
        "AttributeId":"ФИО",
          "Operation": [
          {
                "Date":"2019-01-01 00:00:00",
                "System":"test", "Plan":0,
                "Set":[{"Value":"Иванов И.И."}]
             }
        ]
     },
       {
        "Type":"Literal",
        "AttributeId":"ДатаРождения",
          "Operation": [
          {
                "Date":"2019-01-01 00:00:00",
                "System":"test", "Plan":0,
                "Set":[{"Value":"1979-01-01"}]
             }
        ]
       },
       {
        "Type":"Reference",
        "AttributeId":"РаботаетВ",
          "Operation": [
          {
                "Date":"2019-01-03T00:00:00",
                "System":"test","Plan":0,
                "Set":[{"Value":"ООООбета"}]
             },
          {
                "Date":"2019-01-02T00:00:00",
                "System":"test",
                "Operationid":"918273645",
                "Plan":0,
                "Set":[{"Value":"ОООАльфа"}]
             }
        ]
       }
]
```

```
  }
  ]
 }
}
```

Можно получить расширенную информацию по истории изменения объекта, передав в запросе значение флага **FullDescription** равным 1. В этом случае в теге Operation будут возвращены дополнительные атрибуты:

Request – запрос, выполнение которого привело к данному изменению (например, UpdateObject, DeleteObject, UpdateObjectsGroup и т.п.)
Comment – текстовый комментарий. Содержит текст, переданный в запросе в атрибуте Comment (если такой был) плюс присоединенную мета-информацию:
source - источник поступления запроса (например, *web-form* - через демонстрационную веб-форму; *rest* - через вызов rest-сервиса);
ip – ip-адрес, с которого выполнен вызов (если есть возможность определить).

StorageCode и **StorageName** – код и наименование хранилища, в котором было выполнено изменение.

Transaction – значение атрибута Transaction, если оно было передано в запросе на изменение.

Пример запроса

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectHistory Originator="test" Code="ИвановИИ" FullDescription="1" />
```

Пример ответа

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И. И.">
   <Attribute Type="Literal" AttributeId="ФИО">
    <Operation Date="2019-01-01T00:00:00" System="test"
Request="updateobject" Comment="source:web-form;ip:127.0.0.1"
StorageCode="c4ca4238a0b923820dcc509a6f75849b" StorageName="Fuseki Demo"
Plan="0">
     <Set Value=" Иванов И.И."/>
    </Operation>
   </Attribute>
   …
  </Item>
</Items>
```

5.2.2. Получение истории изменений за период

Запрос позволяет получить список выполненных изменений объектов, отфильтрованый по различным параметрам. Все атрибуты в запросе являются не обязательными.

Параметры

StartDate – дата и время начала периода, за который ищется история изменений.
EndDate – дата и время окончания периода.

Limit - целое число: ограничение на количество возвращаемых объектов, значение по умолчанию 1000.
Offset – целое число, по умолчанию равно 0, количество пропускаемых в результате поиска записей.
FullDescription – вернуть полное описание найденного изменение. Какие свойства не входят в полный набор, указано далее в описании ответа на запрос.
System – код информационной системы, выполнившей изменение.
User - строка, содержащая имя пользователя, выполнившего изменение.
Attribute – атрибут модели, который был изменен.
Code – идентификатор измененного объекта.
Action – произведенное с объектом действие: создание (create), редактирование (update), удаление (delete).
Plan – фактическое (plan=0) или запланированное (plan=1) изменение. По умолчанию возвращаются все виды изменений.
Group – флаг, принимающий значения 0 и 1, по умолчанию 0. Если передано group=1, то результаты будут сгруппированы по объекту, дате и информационной системе.
GroupByCode – флаг, принимающий значения 0 и 1, по умолчанию 0. Если задано значение 1, то в ответ будут возвращены только идентификаторы измененный объектов. Флаги Group и GroupByCode не могут быть заданы одновременно. Если значение 1 задано для обоих параметров, то результат будет как при указании GroupByCode.
Guid – внутренний идентификатор сгруппированного изменения.

ObjectType – тег для задания классов объектов, изменения которых требуется получить. Его атрибуты: **Code** – идентификатор класса.
Item – тег для задания идентификаторов измененных объектов. Атрибуты: **Code** – идентификатор объекта.

Сортировка результатов всегда происходит по дате изменения в обратном порядке.

Ответ

Пакет **History**,

В атрибуте **Count** корневого тега возвращается количество найденных записей. Запись об изменении описывается тегом Operation, содержащим следующие атрибуты и вложенные теги:

Атрибуты, возвращаемые при любых условиях фильтрации в запросе:
Code – идентификатор измененного объекта.
Name – читаемое наименование объекта на момент сохранения изменений. Для мультиязычных данных выводится значение на языке по умолчанию.

Атрибуты, возвращаемые во всех случаях, за исключением задания GroupByCode=1:
Date – дата и время внесения изменения.
Action – выполненное действие: create (создание), update (изменение), delete (удаление).
System – код информационной системы, внесшей изменение.

Plan – 0 или 1 в зависимости от того, является изменение фактическим или запланированным.
Тег *Type* – классы модели, к которым принадлежит измененный объект. Его атрибуты:
Code - идентификатор класса.
Name – читаемое наименование класса (на языке по умолчанию).

Атрибуты и теги, возвращаемые в случае, если не использована группировка:
Attribute – атрибут, значения которого были изменены.

Тег *Set* – набор присвоенных атрибуту значений. Атрибуты тега Set:
Value - присвоенное значение.
Lang – языковая версия. Возвращается только для строковых мультиязычных атрибутов.
Name – читаемое наименование для объекта-значения. Возвращается для значений атрибутов-ссылок. В случае многоязычных данных возвращается только значение на языке по умолчания.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetHistory StartDate="2021-10-19T08:00:00" EndDate="2021-10-19T12:00:00"
Action="update" Limit="100" />
```

В формате JSON:

```
{"GetHistory":{"StartDate":"2021-10-19T08:00:00", "EndDate":"2021-10-
19T12:00:00","Action":"update", "Limit":100}}
```

Пример ответа

В формате XML без использования флага Group:

```
<History Count="3" >
  <Operation Guid="5440F4A4-5A89-4D31-90E8-DF90B5F5B972"
Code="ТестовыйОбьект_1" Date="2021-10-19T11:36:11" Name="Тестовый объект"
Action="update" User="Иванов Петр" System="test" Plan="0"
Attribute="ТестоваяСсылка">
    <Set Value="ТестовыйОбъект_2" Name="Другой тестовый объект"/>
    <Type Code="ТестовыйКласс" Name="Тестовый класс"/>
  </Operation>
  <Operation Guid="5440F4A4-5A89-4D31-90E8-DF90B5F5B972"
Code="ТестовыйОбьект_1" Date="2021-10-19T11:36:11" Name="Тестовый объект"
Action="update" User="Иванов Петр" System="test" Plan="0"
Attribute="Тестовыйтрибут">
    <Set Value="Строка 1" />
    <Type Code="ТестовыйКласс" Name="Тестовый класс"/>
  </Operation>
  <Operation Guid="4B07F376-9B51-4B33-A42C-540D390B7F9F"
Code="ТестовыйОбьект_2" Date="2021-10-19T11:35:26" Name="Другой тестовый
объект" Action="update" System="test_2" Plan="0" Attribute="Тестовыйтрибут">
    <Set Value="Строка 2" />
    <Set Value="Строка 3" />
    <Type Code="ТестовыйКласс" Name="Тестовый класс"/>
  </Operation>
</History>
```

В формате XML при использовании флага Group:

```
<History Count="2" >
  <Operation Guid="5440F4A4-5A89-4D31-90E8-DF90B5F5B972"
Code="ТестовыйОбьект_1" Date="2021-10-19T11:36:11" Name="Тестовый объект"
Action="update" User="Иванов Петр" System="test" Plan="0">
      <Type Code="ТестовыйКласс" Name="Тестовый класс"/>
  </Operation>
  <Operation Guid="4B07F376-9B51-4B33-A42C-540D390B7F9F"
Code="ТестовыйОбьект_2" Date="2021-10-19T11:35:26" Name="Другой тестовый
объект" Action="update" System="test_2" Plan="0">
      <Type Code="ТестовыйКласс" Name="Тестовый класс"/>
  </Operation>
 </History>
```

В формате JSON:


```
{"History": {
      "Count": "3",
      "Operation": [
        {
          "Guid": "5440F4A4-5A89-4D31-90E8-DF90B5F5B972",
          "Code": "ТестовыйОбьект_1",
          "Date": "2021-10-19T11:36:11",
          "Name": "Тестовый объект",
          "Action": "update",
          "User": "Иванов Петр",
          "System": "test",
          "Plan": "0",
          "Attribute": "ТестоваяСсылка",
          "Set": [{ "Value": ТестовыйОбьект_2", "Name": "Другой тестовый
объект" }],
          "Type": [ {"Code": "ТестовыйКласс", "Name": Тестовый класс"}]
        },
        {
          "Guid": "5440F4A4-5A89-4D31-90E8-DF90B5F5B972",
          "Code": "ТестовыйОбьект_1",
          "Date": "2021-10-19T11:36:11",
          "Name": "Тестовый объект",
          "Action": "update",
          "User": "Иванов Петр",
          "System": "test",
          "Plan": "0",
          "Attribute": "Тестовыйтрибут",
           "Set": [{ "Value": "Строка 1" }],
          "Type": [ {"Code": "ТестовыйКласс", "Name": Тестовый класс"}]
        },
        {
          "Guid": "4B07F376-9B51-4B33-A42C-540D390B7F9F",
          "Code": "ТестовыйОбьект_2",
          "Date": "2021-10-19T11:35:26",
          "Name": "Другой тестовый объект",
          "Action": "update",
          "System": "test_2",
          "Plan": "0",
          "Attribute": "Тестовыйтрибут",
          "Set": [{ "Value": "Строка 2" }, { "Value": "Строка 3" }],
          "Type": [{"Code": "ТестовыйКласс", "Name": Тестовый класс"}]
        }
      ]
    }
  }
```


5.2.3. Получение состояния элемента данных на определенную дату

Запросы GetObject и GetObjectsGroup могут быть выполнены с передачей дополнительного параметра Date (тип дата и время) для получения состояния объектов на заданную дату. При этом запрос GetObjectsGroup может исполняться только для хранилищ с нативной поддержкой 4D-историйности, а GetObject – для них и для хранилищ, для которых ведется история значений средствами АрхиГраф. В случае, если хранилище объекта не позволяет получить его состояние в прошлом, будет возвращен InvalidPackage с сообщением об ошибке.

Пример запроса

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObject Originator="test" Code="ИвановИИ" Date="2019-01-01 00:00:00" />
```

Ответ

Пакет **Items** – описание объекта. Значение параметра Date будет возвращено в теге Item ответа.

Пример ответа

В формате XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="ИвановИИ" Name="Иванов И.И." Date="2019-01-01 00:00:00">
   <Type TypeId="Персона" Name="Персоны" />
   <Attribute Type="Literal" AttributeId="ФИО" Value="Иванов И.И." />
   <Attribute Type="Literal" AttributeId="ДатаРождения" Value="1979-01-01"
/>
  </Item>
</Items>
```

5.2.4. Первичная инициализация истории изменения

В случае, если первоначальное заполнение модели и хранилищ данных произошло не через запросы АрхиГраф, для корректной работы с историей изменений требуется первичная инициализация истории. Для этого существуют два служебных метода InitModelHistory и InitObjectsHistory.

InitModelHistory – метод для инициализации истории изменения модели данных. Его единственный и обязательный параметр Date – дата и время, которыми будут помечено первое внесение данных об классах и атрибутах информационной модели.

Пример запроса

В формате XML:

```
<InitModelHistory Endpoint="demo" Originator="system" Date="2019-01-
01T00:00:00" />
```

Пример запроса в формате JSON:

```
{"InitModelHistory": {"Endpoint":"demo", "Originator":"test", "Date":"2019-
01-01T00:00:00"}}
```

В ответ на запрос будет возвращен пакет **OperationResults**.

При повторном вызове метода с той же датой существующие записи о значениях элементов модели за указанную дату будут удалены и заполнены заново.

InitObjectsHistory – метод инициализации истории изменения объектов, принадлежащих указанным классам.

Параметры

Date – дата и время, которыми будет помечена первая история объектов. Обязательный атрибут.

WithoutHistoryOnly – не обязательный, принимает значение 0 и 1, по умолчанию значение 0. Если WithoutHistoryOnly=1, то история будет заполнена только для объектов, для которых она еще не заносилась. Если WithoutHistoryOnly=0 и для объектов уже есть запись за дату, указанную в атрибуте Date, эти записи будут удалены и заполнены заново.

Вложенные теги: ObjectType. Атрибут Code тегов ObjectType содержит код класса, историю для объектов которого требуется инициировать. Задание хотя бы одного тега ObjectType является обязательным.

Пример запроса

В формате XML:

```
<InitObjectsHistory Endpoint="demo" Originator="test" Date="2019-01-
01T00:00:00" WithoutHistoryOnly="1">
  <ObjectType Code="Персона" />
  <ObjectType Code="Компания" />
</InitObjectsHistory>
```

В формате JSON:

```
{"InitObjectsHistory": {
  "Endpoint":"demo",
  "Originator":"test",
  "Date":"2019-01-01T00:00:00",
  "WithoutHistoryOnly":1,
  "ObjectType":[ {"Code":"Персона"}, {"Code":"Компания"} ]
 }}
```

Ответ

Пакет **OperationResults**.

5.2.5. Восстановление объекта на заданную дату

Запрос RestoreObjects позволяет восстановить ошибочно удаленные или измененные объекты. Для восстановления должен быть известен идентификатор объекта или объектов, а также дата и время, на которые объект имел корректные значения.

Пример запроса

В формате XML:

```
<RestoreObjects Endpoint="demo" Originator="system" Date="2019-01-
01T00:00:00">
  <Item Code="ИвановИИ"/>
</RestoreObjects>
```

Пример запроса в формате JSON:

```
{"RestoreObjects": {
  "Endpoint":"demo", "Originator":"test",
"Date":"2019-01-01T00:00:00",
"Item":[{"Code":"ИвановИИ"}]
```

```
}}
```

В ответ на запрос будет возвращен пакет **OperationResults**.

Параметры

Date – дата и время, на момент которых берутся описания восстанавливаемых объектов. Обязательный атрибут.

Тег *Item* - обязателен.

Имеет обязательный атрибут **Code**, содержащий идентификатор восстанавливаемого объекта.

Может возникнуть ситуация, что описание объекта в прошлом не соответствует текущему состоянию модели предметной области. Например, могли быть удалены одни атрибуты и добавлены в качестве обязательных другие; удалены объекты, на которые указывают свойства-ссылки и т.п. В этой ситуации восстановить объект автоматически с помощью RestoreObjects не удастся, и в ответе в OperationResult будет возвращено сообщение об ошибке.

Для восстановления подобных объектов можно поступить следующим образом:

- получить состояние объекта на требуемую дату - запрос getObject с заданием атрибута Date;

- в полученном пакете описания объекта вручную внести требуемые исправления, удалив или добавив значения атрибутов, вызывающие конфликт;

- выполнить изменение объекта с помощью запроса UpdateObject.

Запрос RestoreObjects выполняет фактически те же действия, за исключением пункта о внесении изменения в описание объекта, позволяя упростить восстановление объектов в случае, если коррекция не требуется.

6. REST API

Данный API соответствует стилю REST, чтобы облегчить разработчикам погружение в его детали. Функции, выполняемые этим API, полностью аналогичны функциям основного API, описанным выше.

OpenAPI-спецификация (Swagger) данного интерфейса доступна по адресу https://api.trinidata.ru/swagger/?urls.primaryName=MDM%20API. Данный раздел документации дополняет Swagger-описание, но не заменяет его. Ниже кратко рассматриваются методы этого API и особенности работы с ним.

6.1. Особенности работы GET/POST

Из-за сложности организации модели и из-за того, что идентификаторы сущностей представляют собой URI, при работе с данными сущностями GET-запрос заменяется POST-запросом, PUT без идентификатора сущности заменяет POST-запрос (создание новой сущности). Тело запроса должно представлять собой JSON-документ.

Работа может происходить и только POST запросами – в этом случае метод должен быть указан в самом пакете и иметь название "method".

```
Например:
{
  method: "get",
  ...
}
```

В любом запросе обязательно должны присутствовать следующие переменные:

```
operationId      <= строка, идентифицирующая текущий запрос
systemId         <= идентификатор системы-отправителя запроса
```

Также, если в запросе будет указан параметр rid (request id), то он же будет возвращен и в ответе на данный запрос.

Пример тела запроса:

```
{
  ...,           <= нагрузка запроса
  operationId:   "op1",
  systemId:      "sys1",
  rid:           "req_123"
}
```

ответ:

```
{
  ...,           <= нагрузка ответа
  rid:           "req_123"
}
```

Также могут присутствовать следующие переменные в корне структуры тела запроса:

```
langs: ...       <= языки запроса
method: ...      <= метод запроса
offset: ...      <= смещение на кол-во записей
limit: ...       <= максимальное кол-во возвращаемых записей
order: ...       <= сортировка
rid: ... <= id запроса
```

Параметр method может принимать значения GET/POST/PUT/PATCH.

6.2. Особенности работы PATCH

При наличии конструкции вида "id: null" запрос будет интерпретирован как удаление сущности/ значения свойства с данным id.

при наличии конструкции вида "id: [значение]" запрос будет интерпретирован как добавление сущности/значения свойства с данным id.

6.3. Особенности работы DELETE

В данном API используются DELETE-запросы с телом, а не просто пустые. Существует точка зрения, согласно которой у HTTP DELETE запроса тела быть не может, однако нам

необходимо передавать дополнительные параметры операции удаления. Веб-сервера Nginx и Apache корректно передают тело DELETE-запроса.

Стандартный ответ на запрос DELETE:

```
{
    "rid": "...", <= если был в запросе
    "errors": [],
    "warnings": [],
    "offset": 0,
    "limit": 1000,
    "order": []
}
```

6.4. Многоязычность

Переменные в корне тела запроса:

langs: [lang1, lang2, ...] - языки, на которых запрашиваются значения

либо

langs: "*" - в этом случае будут отданы данные на всех возможных языках

langs: "-" - язык по умолчанию для текущего запроса равен первому указанному аббревиатурой языку в запросе, либо языку по умолчанию точки доступа

Языки – ISO-аббревиатуры из двух букв вида "en", "ru" и т.д.

Все данные, где есть языковые ключи, или сами языки, будут возвращены в алфавитном порядке – это необходимо для того, чтобы возвращаемая структура была всегда постоянной.

Язык по умолчанию – первый язык, указанный аббревиатурой язык в langs, если таковой есть, иначе равен языку по умолчанию точки доступа, к которой делается запрос.

Если в запросе указана "*" – будут возвращены данные на всех языках, если указаны определенные языки – будут возвращены данные только на них.

Языки должны быть отсортированы по алфавиту, если несколько значений для одного языка – они также должны быть отсортированы по алфавиту.

Все названия (для сущностей это значение свойства rdfs:label) отдаются в формате:

```
{
  name: "название на языке по умолчанию, определенному из запроса",
  names: {
          ru: "название на русском",
          en: "название на английском",
          ...
  }
}
```

Где "name" – название на языке по умолчанию, определенному из запроса,

"names" – название на всех языках, определенных из запроса (включая язык по умолчанию, при этом он будет идти первым в перечислении значений свойств объекта).

Если у какого-либо свойства есть несколько значений для одного языка, оно будет возвращено в следующем формате:

```
{
  ru: [ a, b, c, ... ], <= где a, b, c - это значения для одного языка, если
их несколько
  en: [ a2, b2 ... ],
  cn: [ a3 ]
}
```

Т.е. это объект вида "язык": [значения для данного языка].

Если значения какого-либо языка нет, то он не будет присутствовать в объекте вообще.

6.5. Постраничное извлечение данных

Эти свойства всегда присутствуют в ответе, но используются лишь тогда, когда применимы.

```
{
  limit: ...,
  offset: ...
}
```

6.6. Сортировка при запросе

Эти данные присутствуют всегда в ответе, но будут пусты, если не были переданы в запросе.

```
{
  order: {
          id: "asc" / "desc",
          ...
  }
}
```

id – идентификатор свойства, по которому происходит сортировка

asc/desc – направление сортировки

6.7. Формат сообщений об ошибках

Присутствует всегда в ответе API, в случае отсутствия ошибок будет пустой.

```
{
  errors: [
          {
                  msg:    сообщение об ошибке,
                  code:   код ошибки,
                  id:             id сущности, с которой связана ошибка,
```

```
                alias:  alias сущности, с которой связана ошибка,
                exp:    дополнительное, развернутое описание ошибки
        },
        ...
        {
                msg: сообщение об ошибке
        },
        ...
        {
                query:  " ... ",
                msg:    сообщение об ошибке
        }
  ],
  warnings: [
        {
                ...
        },
        ...
  ]
}
```

Если по запросу не найдено ни одного элемента – это не является ошибкой, будет возвращен просто пустой пакет с данными.

В пакетах ошибок могут присутствовать дополнительные вспомогательные элементы, если они необходимы.

не реализовано:

Если ошибка связана с какой-то сущностью, которую можно идентифицировать по id/alias, то они должны присутствовать в ошибке. Если идентификация невозможна, то объект запроса, в котором произошла ошибка, должен быть передан в query.

В ошибках должно быть сообщение о чем-то, что реально является ошибкой (например, элемент не создался, элемент не применим и т.д.). Если же произошло что-то, не являющееся фатальным, это должно передаваться в секции warnings.

Если присутствует хотя бы одна ошибка – HTTP-код ответа должен быть выбран из максимально подходящего к первой ошибке, если нет подходящего - используется 500.

6.8. Правила формирования URL

6.8.1. Базовый URL

Базовый URL REST API имеет форму

```
/ml/[vN]/
```

где vN - версия api, например:

```
/ml/v1/
```

Из данной точки возможно запрашивать то, что может работать без указания точки доступа. Например, запросить список точек доступа можно через GET /ml/v1/endpoints

/ml/ <= означает мультиязычность данных, возвращаемых API

6.8.2. URL с учетом точки доступа

URL с учетом точки доступа (набора данных MDM) имеет форму

/ml/[vN]/endpoints/[endpoint]/

где [endpoint] - id точки доступа

6.9. Получение списка точек доступа

Адрес сервиса:

/ml/[vN]/endpoints

GET/POST, тело запроса:

```
{}
```

ОТВЕТ:

```
{
  items: {
          id: {
                  id:                     id точки доступа,
                  name:           название точки доступа
                  default:        true/false <= является ли точки доступа
точкой доступа по умолчанию
          },
          ...
  }
}
```

6.10. Создание точки доступа

Возможно только для точек-срезов (см. раздел 5.1.1. Создание виртуальной точки доступа), данная точка используется в режиме read-only.

В качестве endpoint в данном запросе должна выступать та точка доступа, срез от которой создается.

PUT, тело запроса:

```
{
  date: "12.05.17 22:45:00" <= дата, на которую создается срез
}
```

ОТВЕТ:

```
{
  item: {
          id:             id новой точки-среза
  }
}
```

Дальнейшая работа не отличается от работы с любой другой точкой доступа.

6.11. Удаление точки доступа

Возможно только для точек-срезов.

В качестве endpoint в данном запросе должна выступать та точка доступа, срез от которой создавался.

DELETE, тело запроса:

```
{}
```

ОТВЕТ:

```
{}
```

6.12. Получение хранилищ

/ml/[vN]/endpoints/[endpoint]/storages

GET/POST, тело запроса:

```
{
  items: [
          {
                  id:     id хранилища,
                  subEntities: true/false
          },
          {
                  entityId: id сущности
          }
  ],
  subEntities:    true/false <= со вложенными сущностями или без, данный флаг
может быть переопределен для определенного хранилища, если указан в items
  на данный момент все хранилища считаются хранилищами с подсущностями
}
```

Если items не указан, то будут возвращены все хранилища данной точки доступа

ОТВЕТ:

```
{
  items: {
          id хранилища: {
                  id:                                   id хранилища,
                  supportHistory:       true/false,
                  supportInference:     true/false,
                  physical:             {
                          type:   тип физического хранилища "mongo/pgsql/...",
                          host:         ...,
                          port:         ...,
                          login:        ...,
                          password:     ...,
                          database:     ...
                  },
                  model:                true/false,
                  readonly:             true/false,
                  multilingual:  true/false,
                  entities:             {
                          id: {
                                  id:           id сущности
                                  name:   название сущности,
                                  physicalTable: ...,
                                  physicalFields: {
                                          id свойства: {
                                                  id:           id свойства
                                                  tableId:id в таблице,
                                                  type:   тип данных*
                                          }
                                  }
                          }
                  }
          }
  }
}
```

*тип данных:

reference – ссылка

xsd:string – строка

xsd:boolean – флаг да/нет (true/false)

xsd:integer – целое число

xsd:double – число

xsd:date – дата

xsd:dateTime– дата и время

6.13. Получение хранилища

/ml/[vN]/endpoints/[endpoint]/storages/[storage]

GET/POST, тело запроса:

```
{
  subEntitites: true/false
}
```

ОТВЕТ:

Аналогично одному значению items из ответа за запрос на получение хранилищ

6.14. Изменение хранилища

/ml/[vN]/endpoints/[endpoint]/storages/[storage]

либо

/ml/[vN]/endpoints/[endpoint]/storages и "id" = [storage] в параметрах

PUT/PATCH, тело запроса:

```
{
  item: {
          ...
  }
}
```

item аналогичен получаемым данным через GET/POST

ОТВЕТ:

```
{}
```

6.15. Формат информации о сущности в ответе API

Под сущностью здесь понимается элемент онтологической модели любого типа – класс, свойство или индивидуальный объект.

Типы сущности перечислены в секции relations, ключ "rdf:type".

Базовые типы сущностей: "owl:Class" / "owl:DatatypeProperty" / "owl:ObjectProperty". Для индивидуальных объектов тип сущности owl:NamedIndividual не указывается, вместо этого передается набор классов модели, которым принадлежит объект.

Описание элемента онтологической модели любого типа представляет собой JSON-объект вида:

```
{
```

```
  id:             id сущности,
  name:   название сущности,
  names: [
          lang1: название на языке lang1 (если названий на данном языке
несколько, то они будут соединены через ","),
          lang2: название на языке lang2,
          ...
  ],
  archive: true/false, <= всегда, даже если не задан (в этом случае он =
false)
  relations: {
          id связи: {
                  id:             id связи,
                  name: название на языке по-умолчанию,
                  names: [
                          lang1: название на языке lang1 (если названий на
данном языке несколько, то они будут соединены через ","),
                          lang2: название на языке lang2,
                          ...
                  ],
                  values: {
                          id на что ссылается: { язык: название сущности* на
что ссылается на данном языке, ... }
                          ...
                  }
          },
          ...
  },
  literals: {
          id: {
                  id:             ...,
                  type:   тип литерала*,
                  name: название на языке по-умолчанию,
                  names: [
                          lang1: название на языке lang1 (если названий на
данном языке несколько, то они будут соединены через ","),
                          lang2: название на языке lang2,
                          ...
                  ],
                  values: {
                          lang: [ val1 ... valN ],
                          ...
                  }
          },
          ...
  }
}
```

тип литерала* = xsd:integer/xsd:date/xsd:string/...

6.16. Формат информации о сущности в запросе к API

Можно передать описание сущности точно в таком же виде как в описанном выше ответе.

Однако, есть дополнительные варианты для упрощения.

Так, relations могут быть описаны в таком виде:

значение свойства будет полностью замещено новым значением

```
{
  id связи: значение
}
```

или

значение свойства будет полностью замещено новыми значениями

```
{
  id связи: [ значение1, значение2, ... ]
}
```

или

значение свойства будет полностью замещено новыми значениями

```
{
  id связи: {
          id: id связи, <= опционально,
          values: [ значение ]
  }
}
```

или

удаление всех значений свойства

```
{
  id связи: {
          id: id связи, <= опционально,
          values: null <= удалить все такие связи
  }
}
```

или

частичное замещение/удаление/добавление значений свойства

```
{
  id связи: {
          id: id связи <= опционально,
          values: {
                  id1: null, <= эту связь удалить
                  id2: true <= эту добавить/оставить
          }
  }
}
```

Секция literals может иметь вид:

полное замещение значений свойства для всех языков, значение будет добавлено на текущем языке

```
{
  id: значение, <= для языка по умолчанию для текущего запроса, тип нужно
будет находить МДМу
}
```

или

полное удаление значений свойства для всех языков

```
{
  id: null
}
```

полное удаление значений свойства для всех языков

```
{
  id: {
          id:             значение, <= опционально
          type:   тип, <= опционально
          values: null <= удалить данный литерал
  }
}
```

или

полное замещение значений свойства для текущего языка

```
{
  id: {
          values: [ значение1, значение 2, ... ] <= значения для текущего
языка запроса
  }
}
```

или

замещение/удаление/добавление значений свойства для указанного языка

значения для конкретного языка всегда замещают все его значения (т.е. нельзя добавить/удалить конкретное значение конкретного языка)

```
{
  id: {
          values: {
                  "-":    null <= удалить все значения для текущего языка
запроса
                  "ru":   [ значение ] <= значения для языка "ru",
                  "en":   null <= удалить все значения для языка "en"
          }
  }
}
```

6.17. Запрос сущностей

/ml/[vN]/endpoints/[endpoint]/entities

Запрос одной сущности:

POST, тело запроса:

```
{
  items: [ id ]
}
```

Запрос нескольких сущностей:

POST, тело запроса:

```
{
```

```
  items: [ id1...idN ]
}
```

или

POST, тело запроса:

```
{
  classes: {
          "items" => [
                  id1,
                  id2,
                  ...
          ]
  }
}
```

ОТВЕТ:

см. выше

6.18. Удаление сущностей

/ml/[vN]/endpoints/[endpoint]/entities

Удаление одной сущности:

DELETE, тело запроса:

```
{
  items: [ id ]
}
```

Удаление нескольких сущностей:

DELETE, тело запроса:

```
{
  items: [ id1...idN ]
}
```

ОТВЕТ:

см. выше

6.19. Создание/замещение сущностей

PUT, тело запроса:

```
{
  items: [
          {
```

```
                alias: "уникальное название для данной сущности в пределах
данного запроса",
                id: "идентификатор сущности"
                ...
        }
  ]
}
```

Если указан id, и сущность с таким id уже существует, то выполняется полное замещение всех значений ее свойств на переданные.

Если указан id, но данной сущности нет – будет создана сущность с таким id и переданными значениями свойств.

Если id не указан – он будет автоматически сгенерирован (это предпочтительный вариант работы).

alias – псевдоним для данной сущности внутри текущего запроса, аналог LocalCode стандартного API внутри запроса.

Если alias не указан, но указан id, полагается alias = id.

При наличии каких-либо ссылок внутри текущего запроса (у данной сущности или у любой другой) со значением = alias, вместо данное значения должен быть передан id этой сущности.

6.20. Изменение сущностей

Возможно только для существующих сущностей.

PATCH, тело запроса:

```
{
  items: [
        {
                id: "идентификатор сущности"
                ...
        }
  ]
}
```

Работает полностью аналогично PUT за исключением того, что все сущности, упомянутые в запросе, обязаны существовать в MDM.

Происходит не полная, а частичная замена сущностей. Однако для литералов на указанном языке будет происходить полная замена.

Если для ссылки или литерала будет передан null, это означает, что все значения соответствующей ссылки или литерала для данного языка/всех языков будут очищены.

Пример:

```
{
  items: [
        {
                id: "firm1",
```

```
                relations: {
                        "rdf:type": {
                                ...
                        },
                        "city": {
                                id:             "city" <= опционально,
                                values: {
                                        "moscow":       null, <= удаление
ссылки "id" со значением "moscow"
                                        "sochi":        true, <= добавление
ссылки "id" со значением "sochi"
                                }
                        },
                        "related": {
                                id:             "related",
                                values: [ "abc1", "abc2", ... ]
                        },
                        ...
                },
                literals: {
                        id: null,
                        id: {
                                id: ...,
                                type: ...,

                                values: {
                                        lang1:  null,
                                        "-":    [ "123" ]
                                }
                        },
                        ...
                }
        }
  ]
}
```

6.21. Фильтрация при запросе сущностей

Если нужно получить сущности определенного типа, необходимо указать фильтр по свойству с идентификатором "rdf:type".

```
filters: {
  combine: "or/and", <= объединение группы фильтров
                                        OR - если один из фильтров группы
сработал, значит элемент прошел проверку
                                        AND - если хотя-бы один из фильтров
не сработал, значит элемент не прошел проверку
  items: [
        {
                combine: "or/and",
                items: [
                        {
                                id сущности*: {
                                        comparison: операция сравнения**
                                        value: значение***
                                },
                                ...
                        }
                        ...
                ]
        },
```

```
            {
                    combine: "or/and",
                    items: [ ... ]
            }
    ]
}
```

* id сущности может быть как реальным id сущности, так и значением alias

** операции сравнения, зависят от типа сравниваемого значения.

Для типов strings/numbers/references/dates:

equal - требует value, =, все типы

notEqual - требует value, !=, все типы

exists - не требует value, проверка существования, все типы

notExists - не требует value, проверка несуществования, все типы

Для типов numbers/dates:

less - требует value, <, все типы кроме строк

lessEqual - требует value, <=, все типы кроме строк

more - требует value, >, все типы кроме строк

moreEqual - требует value, >=, все типы кроме строк

Для типов strings:

contains - требует value, проверка на содержание подстроки в строке (регистрозависимая), только строки

iContains - требует value, проверка на содержание подстроки в строке (регистронезависимая), только строки

*** значение - должно быть только у тех операций сравнения, у которых оно требуется

Указание в запросе на извлечение списка классов, которым принадлежит сущность:

```
classes: {
  combine: "or" / "and",
  items: []
}
```

6.22. Получение модели

/ml/[vN]/endpoints/[endpoint]/schema

POST, тело запроса:

```
{
  items: [
          {
                  id:                                                id сущности,
с которой происходит запрос схемы, может отсутствовать
                  subEntities:                    true/false - включать
подсущности, по умолчанию true
                  entitiesInheritance:   true/false - включать наследуемые
сущности, по умолчанию false
                  expandRanges:                   true/false - включать в
описание ссылок все возможные сущности, по умолчанию true
                  entities:                                 true/false - включать
в ответ связи и литералы для классов, по умолчанию true
          }
  ]
}
```

ranges – диапазон возможных значений

domains – классы, объекты которых могут обладать значениями этого свойства (вверх по иерархии классов)

ОТВЕТ:

```
{
  "items": [
          {
                  id:                                                id/null,
                  subEnitites:                    true/false,
                  entitiesInheritance:   true/false,
                  expandRanges:                   true/false,
                  entities:                                 true/false,

                  items: {
                          id: {
                                  id:                     id сущности,
                                  name:           название сущности,
                                  names:          [ названия сущности на разных
языках ]
                                  archive:        true/false,
                                  type:           owl:class/...,
                                  rangesOf:       [ id, ... ].
                                  domainsOf:      [ id, ... ],
                                  parents:        [ id, ... ],
                                  ranges: {
                                          operator:       or/and, <= режим
объединения id сущностей из items
                                          items:          [ id, ... ] <= id
сущностей
                                  }
                                  domains: {
```

```
                                        operator:       or/and, <= режим
объединения id сущностей из items
                                        items:          [ iid, ... ] <= id
сущностей
                                }
                                minCardinality: ..., <= минимальное
количество значений для данной сущности, если >= 1 - указанное количество
значений обязательно, null - не задано
                                maxCardinality: ..., <= максимальное
количество значений для данной сущности, null - не задано
                        },
                        ...
                }
        }
  ]
}
```

rangesOf – список id свойств, значениями которых могут быть объекты данного класса

domainsOf – список id свойств, значениями которых могут обладать объекты данного класса"

parents – список id сущностей, которым непосредственно подчинена (в которые непосредственно вложена) данная сущность

ranges – список id классов, объекты которых могут быть значениями данного свойства

domains – список id классов, объекты которых могут обладать значениями данного свойства

6.23. Работа с историей изменений сущностей

/ml/[vN]/endpoints/[endpoint]/history/entities/

POST, тело запроса:

```
{
  items: id сущности
}
```

или

```
{
  items: [ id1, id2, ... ]
}
```

или

```
{
  items: [
          {
                  id: ...,
                  dateFrom: ...,
                  dateTo: ....
          }
  ]
}
```

limit и offset работают на данном запросе на каждый переданный item

ОТВЕТ:

```
{
  items: [
          {
                  id сущности: {
                          id: id сущности,
                          operations: [
                                  {
                                          id: id сущности,
                                          date: дата совершения операции,
                                          type: "create"/"delete"/"modify",
                                          systemId: id пользователя,
совершившего операцию,
                                          operationId: id операции,
                                          properties: {
                                                  relations: {
                                                          id сущности
(свойства): [
                                                                  {
                                                                          id:
  id сущности (свойства),
                                                                          name:
  название сущности (свойства),
                                                                          names:
[ ... ]
                                                                  },
                                                                  ...
                                                          ],
                                                          id сущности
(свойства): null <= была полностью удалена
                                                          ...
                                                  },
                                                  literals: {
                                                          id сущности
(свойства): {
                                                                  язык: [
значение1, значение2, ... ],
                                                                  ...
                                                          },
                                                          id сущности
(свойства): null <= была полностью удалена,
                                                          id сущности
(свойства): {
                                                                  язык: null <=
была удалена на определенном языке
                                                          }
                                                          ...
                                                  }
                                          }
                                  },
                                  ...
                          ]
                  }
          ]
  ]
}
```

Классы, которым принадлежит сущность, передаются как значения свойства rdf:type.

6.24. Получение конкретной подписки

/ml/[vN]/endpoints/[endpoint]/subscriptions

POST, тело запроса:

```
{
  items: id подписки
}
```

или

/ml/[vN]/endpoints/[endpoint]/subscriptions/[id]

или

получить конкретные подписки

```
{
  items: [ id1, id2, ... ]
}
```

или

получить все подписки для данной системы

```
{}
```

ответ

```
{
  items: {
          id подписки: {
                  id: id подписки,
                  entitiesIds: [ ... ],
                  entitiesExcludeIds: [ ... ],
                  deferred: ...,
                  type: ...,
                  enabled: ...,
                  physical: {
                          type:           ...,
                          host:           ...,
                          port:           ...,
                          login:          ...,
                          password:       ...,
                          queue:          ...
                  }
          }
  }
}
```

6.25. Создание подписки

PUT, тело запроса:

```
{
  "item": {
          entities:               [ ... ],
          entitiesExclude:[ ... ],
          deferred:               true/false, по умолчанию true (изменения не
отсылаются мгновенно),
          type:                   "all" / "individuals" / "scheme", по
умолчанию "all" <= на что подписка
          enabled:                true/false, по умолчанию true <=
включение/отключение подписки
          physical: {
                  type:           "rabbitmq"/"kafka"/..., <= тип очереди
                  host:           url сервера очереди,
                  port:           порт сервера,
                  login:          имя пользователя,
                  password:       пароль пользователя,
                  queue:          id очереди
          }
  }
}
```

ОТВЕТ:

```
{
  item: {
          id: id подписки
  }
}
```

6.26. Изменение подписки

PATCH, тело запроса:

```
{
  item: {
      id: ...,
          enabled: true/false
  }
}
```

ОТВЕТ:

```
{
}
```

6.27. Удаление подписки

DELETE, тело запроса:

```
{
  item: {
          id: ....
```

```
  }
}
```

или

```
{
  item: id
}
```

ОТВЕТ:

```
{}
```

6.28. Сброс кэша

/ml/[vN]/endpoints/[endpoint]/cache

GET/POST/DELETE, тело запроса:

```
{
  items: {
          cacheId1: {
                  id: cacheId1,
                  entities: [ id1, id2, ... ] <= конкретные сущности для
удаления из кэша
          },
          ...
  }
}
```

либо

```
{
  items: [ cacheId1, cacheId2, .... ]
}
```

либо

```
{
  items: [
          {
                  id: cacheId1,
                  entities: [ id1, id2, ... ] <= конкретные сущности для
удаления из кэша
          },
          ...
  ]
}
```

cacheId может принимать значения:

schemas - модель данных

entities - сущности

langs - языки

endpoints - точки доступа

all - все вместе