ArchiGraph platform

API Reference

Table of contents

1. ArchiGraph platform purpose

The ArchiGraph platform is intended for storing an information model, normative and reference information, basic and transactional data of an organization. It provides access for the application software components both to data located in storages under the platform's control, and to any information in external storages available to the platform.

The ArchiGraph platform is intended for use as the core of complex applied automated systems, including:

- situation centers,
- integrated analytical systems and logical data marts,
- reporting systems, etc.

ArchiGraph can also act as a tool of corporate data management (a data governance tool) in the complex, integrated infrastructures consisting of dozens of business applications.

ArchiGraph provides data synchronization between different applications, access rights control, model and data versioning, data extraction from external sources in a logical mart mode, notifications by subscription about changes in the model and data, data integrity control / logical validation and many other functions.

The data model of ArchiGraph is presented in the form of an ontological model. It allows client applications to work with the machine-readable representation of data as well as its structure. Ontologies offers faceted classification, attributes set inheritance, multiple values of each attribute of each object, multi-language attributes values etc.

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. Data exchange architecture

2.1. Query formats

ArchiGraph API is based on the exchange of packages with other software components. Two main request formats are supported:

- JSON
- XML

It is also possible to read data platform using SPARQL and GraphQL query languages.

2.2. Transport

XML or JSON packets transport can be performed with a number of ways:

- using RabbitMQ or Kafka queues (the preferred method, and the only one possible in high-load and highly available architectures),

- REST-services,
- Websocket protocol,
- web-interfaces (for debugging).

Packet formats and the sequence of their exchange do not depend on the transport. During the exchange process, each application that wants to send a request to ArchiGraph must generate a message in any of the supported formats and send it to the system using the appropriate transport. The platform will return the response using the same transport protocol. The outgoing packet will be returned in the same format as the incoming one.

3. ArchiGraph platform management tools

ArchiGraph platform has a command-line management tool named "mdmctl". It allows management of the following platform settings:

- Data storages
- Endpoints (data sets)
- Client software components
- Data exchange queues
- Subscriptions
- Constants
- Standard-defined classes and attributes.

Its use is covered in the “Mastering corporate information systems with ArchiGraph platform: a cookbook” brochure.

There is also a special web tool for data edit requests moderation.

The ontology structure can be managed by the analyst using the visual web-based ontologies editor ArchiGraph.Mir. Some tools for working with platform data are also built into the ArchiGraph.KMS (knowledge management system) product. These components provide the following functions:

- Ontology editing interface allows you to create, edit, delete classes, attributes and object instances, view them as trees or lists with sorting and filtering capabilities, find them by quick search by substring (part of the name), set values for any properties of any objects (including several values for each attribute, if allowed by the information model constraints), as well as attach files and user comments to them.
- ArchiGraph.KMS provides creation and applying of duplicate objects searching rules according to the conditions designed by the analyst. A special interface is provided for viewing and merging duplicate objects (data normalization).
- Collaborative ontology editing and access rights management. The access rights management tool for provides an administrator with the opportunity to select a user group and set access rights for all or some classes of the ontology classes hierarchy to one of the following levels: no access, read only, edit with confirmation, edit.
- The information model editing interface takes into account the user's access rights to the model, displays an error message when an attempt is made to perform an operation that is not allowed by the current user.

- When editing the ontology and the data represented according to it, a user with "edit with confirmation" rights to any data item has the opportunity to create a request to change this record. The moderator (a user with full editing rights) interface allows to view requests for changing data items, approve or reject each request.
- The editor supports import / export of model fragments via Excel files.
- The editor logs all changes to the ontology made by different users and provides an opportunity to view changes history.

ArchiGraph platform offers a simple web form for debugging and testing its API (see Fig. 1).

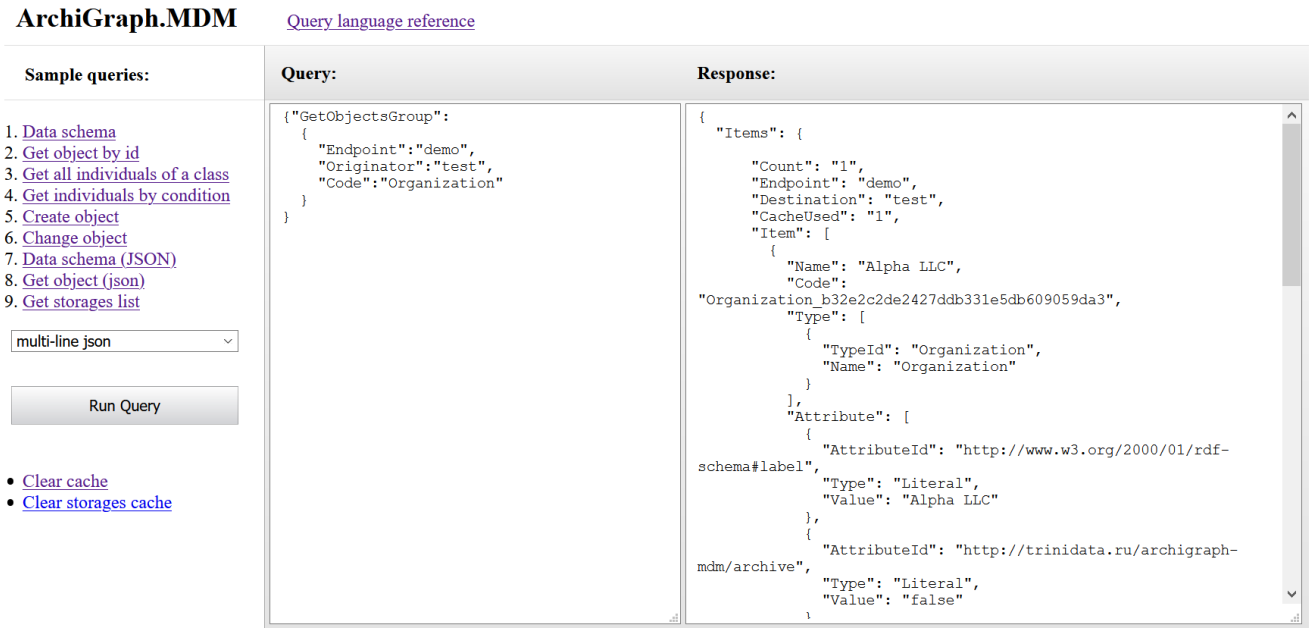

Fig. 1. Web form for ArchiGraph API queries testing

On the left thera are links to sample queries. Clicking on any link copies the request text into the "Query" field, where you can change it. Clicking on the "Run query" button performs a request to the platform, the answer is displayed in the "Response" field.

To make HTTP requests to the system from the software components, you need to send a POST request to the URL /mdm, passing the body of the XML or JSON request in the "request" parameter.

4. ArchiGraph platform core API

4.1. General information

XML and JSON packets have the similar structure.

An example of XML packet:

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

The same packet in JSON format:

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

Tag and attribute names are case insensitive.

Any requests, regardless of the root tag, can have the following standard parameters:

Endpoint – code of the endpoint[1] to which the request is addressed. It is applicable to all queries except for the request to get a list of endpoints (GetEndpoints). This parameter is optional: if the access point code is not specified, the request will be routed to the endpoint defined by the system settings as the default endpoint. If a parameter value is passed in an input packet, then that value will be returned in the root response tag in the Endpoint attribute.

OperationId – query identifier. Is it applicable to all queries. If an OperationId is passed in an input packet, then its value will be returned in the OperationId attribute of the root response tag. It is optional, but it is necessary when exchanging packets through message queues, as it allows you to match an outgoing request with its response.

Originator – the code of the requesting client component/system. Is it applicable to all queries. If a parameter is passed in a request, then its value will be returned in the Destination attribute of the root response tag. The component/system with the transferred code must be registered in the platform and the access rights must be configured for it. In general, the Originator parameter is not mandatory, but if requested without specifying the system, the lowest access rights will be applied (access rights for anonymous systems).

Token – applicable to all queries. The string serves to ensure information security: to identify the system, the correspondence of the system code transmitted in the Originator attribute with the token registered for it is checked. An information system can be registered with an empty token, in this case the Token parameter does not need to be passed.

Let us note that this token is constant and defined at the platform configuration. External authorization service (such as KeyCloak) also may be used, to protect ArchiGraph API with the JSON web tokens mechanism.

User – applicable to all queries. Optional string parameter. It is recommended that you pass the name of the user whose action in the application caused this request to be sent as the value of the User parameter. The passed value is for reference only, it will be recorded in the platform logs and the data changes history but will not affect the execution of the request.

Comment – applicable to all queries. Optional parameter: an arbitrary text comment to the request. The passed value will be stored in the MDM logs and does not affect the execution of the request.

If an error occurs while processing any request, the framework returns an **InvalidPackage** in response.

Пакет **InvalidPackage** – a message informing on the erroneous query. InvalidPackage has the following attributes:

Message – error message text

[1] An endpoint is a separate part of the logical data space. Endpoints are somehow similar to the databases in a relational DBMS.

XML packet example:

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

JSON packet example:

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

4.2. Endpoints and storages configuration queries

4.2.1. Get the set of endpoints

GetEndpoints – a request for the list of endpoints accessible for the requesting client.

`Parameters`

No specific extra parameters.

`Sample query`

XML format:

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

JSON format:

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

`Response`

Endpoints packet

Nested tags: *Endpoint*

Endpoint tag contains the description of one endpoint.

Endpoint tag attributes:
Code – endpoint code. Used as the value of Endpoint parameter of the queries;
Name – readable name of an endpoint;
Default – true if this endpoint is default, false otherwise.

`Example`

XML format:

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

JSON format:

```
{
   "Endpoints":
    {
    "Destination": "test",
    "Endpoint": [
        {
            "Code":"demo",
```

```
            "Name":"Demo endpoint",
            "Default":"true"
        }
      ]
    }
  }
```

4.2.2. Get storages list

GetStorages – request to obtain a list of storages with their characteristics: a set of the classes which individuals are stored in it, data changes history support, temporality support, operation in logical data mart (LDM) mode, type of physical storage.

Parameters

If the **EndpointCode** attribute is passed in the request, a list of storages related only to the specified endpoint will be returned. If the endpoint code is not specified, the storages of all points available to the information system will be returned.

For each storage, its Parameters (code, name, access details, etc.) are returned, as well as a list of model classes whose objects are contained in the specified storage. By default, only top-level classes will be returned, the **WithSubclasses** attribute allows you to get a complete list of classes, including inherited ones. The WithSubclasses attribute takes values 0 and 1, the default is 0.

Sample query

XML format:

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

JSON format:

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

Response

Storages packet

Nested tags:

Storage – contains the storages list

ObjectType – contains the ontology class which individuals are stored in this storage

Storage tag

Storage tag attributes:

Code – storage code (identified) in the platform
Name – storage readable name

Endpoint – the code of an endpoint bound with this storage
History – boolean, takes the values 0 and 1. Shows whether the data changes history is stored by the ArchiGraph.
Timestamp – boolean, takes the values 0 and 1. Determines whether the storage itself supports full data history (temporality support). Such support is provided, for example, in HBase by storing a set of values for each attribute of an object, and each value is marked with a timestamp.
Logic – boolean, takes the values 0 and 1. Determines whether the storage supports inferencing.
Type – physical storage type (PostgreSQL, MongoDB etc)
Model – boolean, takes the values 0 and 1. Determines if this storage contains TBox, the definitions of the ontology classes and properties. The model storage is also a default individuals storage which contains individuals of the classes not bound explicitly to any other storage.
Editable – boolean, takes the values 0 and 1. Determines if the storage is editable, i.e. does it support creating/editing/deleting objects with the ArchiGraph platform API requests.
Multilang – boolean, takes the values 0 and 1. Determines if the storage support multilingual data.

ObjectType tag attributes:

Code – ontology class identifier
Name – readable class name

`Response example`

XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Storages Destination="test">
  <Storage Code="storage1" Name="TDB TBox storage" Endpoint="demo"
History="1" Timestamp="0" Logic="1" Type="Fuseki" Editable="1" Multilang="1"
/>
  <Storage Code="storage2" Name="Customers database" Endpoint="demo"
History="0" Timestamp="0" Logic="1" Type="MongoDB" Editable="1"
Multilang="1">
     <ObjectType Code="Customer" Name="Customers" />
  </Storage>
</Storages>
```

JSON format:

```
{"Storages:
  {
    "Destination":"test",
    "Storage": [
      {
        "Code":"storage1",
        "Name": "TDB TBox storage",
        "Endpoint": "demo",
        "History":"1",
        "Timestamp":"0",
        "Logic":"1",
        "Type": "Fuseki",
        "Editable":"1",
        "Multilang":"1"
```

```
        },
        {
          "Code":"storage2",
          "Name": "Customers database",
          "Endpoint": "demo",
          "History":"0",
          "Timestamp":"0",
          "Logic":"1",
          "Type":" MongoDB",
          "Editable":"1",
          "Multilang":"0"
          "ObjectType": [
            {"Code":"Customer", "Name":"Customers"}
          ]
        }
      ]
    }
  }
```

4.2.3. Storage parameters update

UpdateStorage – storage parameters update request

`Parameters`

Nested tags: *Storage*.

Storage tag attributes:

Code – mandatory, the code of an updated storage
Name – optional, string. Readable name of the storage
History – optional, 0 or 1. Start or cancel objects properties history recording.
AddTypes – optional, 0 or 1. If 0 or an attribute not set, then the list of the classes bound to the storage will be replaced with the list of the classes defined in the ObjectType tag. If 1, the list of the classes bound to this storage will be extended with the given classes.
DeleteTypes – optional, 0 or 1. If 1 is set, the classes listed in the ObjectType tag will be removed fro the list of classes bound to this storage.

Nested tags: *ObjectType*

ObjectType tag attributes:

Code – URI of the ontology class

`Sample query`

XML format:

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

`Response`

OperationResults – a packet describing the operation results. See its description in the UpdateObject request definition.

4.2.4. Get storage associated with class

GetStoragesMapping – get information on the storage where individuals of the given class are stored: database type and connection requisites, table/collection name, attributes to column names mapping. The result can be used for the direct storage access: for performing MapReduce tasks etc.

Parameters

ObjectType nested tag – the list of the ontology classes

ObjectType tag attributes:

Code – mandatory, ontology class identifier

Sample query

XML format:

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

JSON format:

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

Response

MapStorages packet

Nested tags:

MapStorage – description of the storage containing objects of the given class,

MapAttribute – description of the ontology property – to – table column mapping,

Target – for the references (ObjectProperty) – the range of an attribute.

MapStorage tag attributes:

ObjectType – ontology class URI
Code – ArchiGraph storage code
Name – readable name of the storage
Type – storage database type
Host, **Port**, **Login**, **Password**, **Database** – database connection requisites (note: in the base version ArchiGraph does not return database login and password – if needed, this shall be customized in the particular platform installation)
Table – name of the table or collection which stores individuals of a given class

MapAttribute tag attributes (nested in MapStorage):

AttributeId – ontology property URI

Field – field or column for storing values of the property
Type – kind of property: Literal (for DatatypeProperty) or Reference (for ObjectProperty)
Datatype – data type (range) for literal attributes. Possible values:

- xsd:string
- xsd:boolean
- xsd:integer
- xsd:double
- xsd:date
- xsd:dateTime

You can get all the literal data types supported by platform by requesting all individuals of archigraph:type class.

XML format request for all the literal data types:

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

The same request in JSON format:

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

Target tag attributes (nested in MapStorage):

Code – class URI

Name – class readable name

Response example

XML format:

```
 <MapStorages Endpoint="demo" Destination="test">
  <MapStorage ObjectType="Customer" Code="storage2" "Name"="Customers
database" Type="MongoDB" Host="12.12.12.123" Port="27017" Database="demo"
Table="clients">
    <MapAttribute AttributeId="VATCode" Field="vat" Type="Literal"
Datatype="xsd:string" />
    <MapAttribute AttributeId="hasCity" Field="city" Type="Reference">
    <Target TargetId="City" Name="Cities" />
  </MapAttribute>
  </MapStorage>
  <MapStorage ObjectType="City" Code="storage1" Type="Fuseki"
Host="12.12.12.123" Port="8080" Table="demo" />
</MapStorages>
```

4.3. TBox queries

4.3.1. Get the classes and properties

GetDataSchema – request for data model description, containing list of classes and properties.

Parameters

StartElement – the starting class of the requested branch of hierarchy. If omitted, all the model will be returned.

WithoutSubClasses – optional attribute, can be 0 or 1, 0 by default. Only meaningful when you specify StartElement. If the attribute value is 0 or not specified, then the description of the model is returned both for the class specified in the StartElement attribute and for all its subclasses. If WithoutSubClasses = 1, then only the description of the class specified in StartElement is returned.

WithoutInherited – optional attribute, can take the values 0 and 1, the default value is 0. If the value is 0 or not specified, the class description contains all the attributes applicable to objects of this class, including attributes inherited from parent classes. If WithoutInherited = 1, then for each class only attributes associated directly with the specified class will be returned, without inherited ones.

WithoutRangeInherited - optional attribute, can take the values 0 and 1, the default value is 0. If the value is 0 or not specified, then the range of the reference properties includes a complete list of classes, including subclasses. If WithoutInherited = 1, then only classes associated directly with the attribute will be returned for the range of the reference property, without nested ones.

WithoutAttributes – optional attribute, can take the values 0 and 1, the default value is 0. When setting WithoutAttributes, the description of classes without a list of attributes will be returned.

Sample query

XML format:

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

Sample query JSON format:

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

Response

DataSchema – the packet containing information model (TBox) description: a list of the classes and properties.

DataSchema packet attributes

StartElement – the root element of the returned model fragment, if set in the request

Prefix – the default prefix of the ontology elements. In all the other responses prefix is omitted, if matches default prefix. Several prefixes can be used in one ontology. In this case, the identifiers of the model elements having non-default prefix will be returned in the full URI form: http://.../.../[element id].

ObjectType tag – describes one class

ObjectType attributes:

Code – class URI

Name – class readable name

Archive – boolean, true or false. Set true for the classes marked as obsolete.

Parent tag – describes superclasses of this class. A class can have several superclasses.

Parent tag attributes:

ParentId – parent class URI

Attribute tag

Describes an attribute applicable to the individuals of the described class. Attributes applicability is inherited from superclasses to subclasses, i.e. if "VAT code" attribute is applicable to the "Formal organization" class, then it is also appliable to its "Company" subclass.

Attribute tag attributes

AttributeId – attribute URI

Name – attribute readable name

Type – attribute kind: Literal for DatatypeProperty attributes, Reference for ObjectProperty attributes.

DataType – is used for the literal attributes. Shows property range as one of the xsd datatypes: integer, string, date, dateTime, boolean, double.

MinCardinality – a minimal number of this attribute's values for every individual. If MinCardinality =1, the attribute is mandatory.

MaxCardinality – a maximal number of this attribute's values for every individual. MinCardinality and MaxCardinality may be omitted, then attribute can take unlimited number of values for every individual.

RangeIntersection – optional. It is set for the reference attributes if the range of values is the intersection of several classes (RangeIntersection = "true"). By default, if the RangeIntersection attribute is not specified, the range is assumed to be the union of the classes specified in the nested Target tags.

ClassSet – optional attribute, can be set for literal properties and reference properties. Used when the scope of the property is the intersection of several classes. In this case, the ClassSet value specifies (separated by commas) the identifiers of those classes to which the object must simultaneously belong in order for the described property to be applicable to it.

Archive – can be set to true or false. True corresponds to deprecated attributes, the description of which has been preserved for backward compatibility of the accumulated data and the current model.

Target tag

Specifies the classes whose individuals can be values for a given reference attribute.

Target tag attributes:

TargetId – class URI

Name – class readable name

Response example

XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSchema StartElement="Company" Prefix="http://some-model.ru/instance">
 <ObjectType Code="Company" Name="Companies">
  <Parent ParentId="FormalOrganisation"/>
  <Attribute Type="Literal" AttributeId="hasPhone" Name="Phone number"
DataType="xsd:string" MinCardinality="1" />
 </ObjectType>
 <ObjectType Code="Person" Name="Persons">
  <Parent ParentId="Agent"/>
  <Attribute Type="Literal" AttributeId="hasBirthDate" Name="Birth date"
DataType="xsd:date" MinCardinality="1" MaxCardinality="1"/>
  <Attribute Type="Reference" AttributeId="worksIn" Name="Works in">
    <Target TargetId="Company" Name="Companies" />
  </Attribute>
 </ObjectType>
</DataSchema>
```

JSON format:

```
{"DataSchema": {
 "StartElement":"Company",
 "Prefix": "http://some-model.ru/instance",
 "ObjectType": [
  {
    "Code": "Company",
    "Name": "Companies",
    "Parent": [ {"ParentId": "FormalOrganisation"} ],
    "Attribute": [
      {
        "Type": "Literal",
        "AttributeId": "hasPhone",
        "Name": "Phone number",
        "DataType": "xsd:string",
        "MinCardinality":1
      }
    ]
  },
  {
    "Code": "Person",
    "Name": "Persons",
    "Parent": [ {"ParentId": "Agent"} ],
    "Attribute": [
      {
        "Type": "Literal",
        "AttributeId": "hasBirthDate",
        "Name": "Birth date",
        "DataType": "xsd:date",
        "MinCardinality":1,
        "MaxCardinality":1
      },
```

```
        {
          "Type": "Reference",
          "AttributeId": "worksIn",
          "Name": "Works in",
          "Target": [ {"TargetId": "Company", "Name": "Companies"} ]
        }
      ]
    }
    ]
  }
}
```

4.3.2. Get the classes and properties in compact form

GetDataSchemaCompact – request for data model description, containing list of classes and properties in the compact form.

Parameters

StartElement – the starting class of the requested branch of hierarchy. If omitted, all the model will be returned.

WithoutSubClasses – optional attribute, can be 0 or 1, 0 by default. Only meaningful when you specify StartElement. If the attribute value is 0 or not specified, then the description of the model is returned both for the class specified in the StartElement attribute and for all its subclasses. If WithoutSubClasses = 1, then only the description of the class specified in StartElement is returned.

WithoutInherited – optional attribute, can take the values 0 and 1, the default value is 0. If the value is 0 or not specified, the class description contains all the attributes applicable to objects of this class, including attributes inherited from parent classes. If WithoutInherited = 1, then for each class only attributes associated directly with the specified class will be returned, without inherited ones.

Sample query

XML format:

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

JSON format:

```
{"GetDataSchemaCompact": {"StartElement":"Company"}}
```

Response

DataSchemaCompact packet – data model description in the compact form

This packet is similar to the above described DataSchema package, except that the list of applicable properties is not repeating for each class. Instead of it, the set of properties is defined by the separate AttributeDefinition tags, and for each class the list of properties applicable to its individuals is listed in the ApplicableAttribute tags (inside ObjectType tag).

AttributeDefinition tag – describes property applicable to the individuals of some class of ontology.

AttributeDefinition tag attributes:

AttributeId – attribute URI

Name – attribute readable name

Type – attribute kind: Literal for DatatypeProperty attributes, Reference for ObjectProperty attributes.

DataType – is used for the literal attributes. Shows property range as one of the xsd datatypes: integer, string, date, dateTime, boolean, double.

MinCardinality – a minimal number of this attribute's values for every individual. If MinCardinality =1, the attribute is mandatory.

MaxCardinality – a maximal number of this attribute's values for every individual. MinCardinality and MaxCardinality may be omitted, then attribute can take unlimited number of values for every individual.

Archive – can be set to true or false. True corresponds to deprecated attributes, the description of which has been preserved for backward compatibility of the accumulated data and the current model.

Target tag

Defines classes range which individuals can be the values for the reference property.

Атрибуты тега Target

TargetID – class URI

Name – class readable name

ApplicableAttribute tag

Indicates that the individuals of the described class can have values of this property (property domain).

ApplicableAttribute tag

AttributeID – property URI.

Response example

XML format:

```
<DataSchemaCompact Destination="test" Prefix="http://trinidata.ru/demo/"
StartElement="FormalOrganisation">
  <AttributeDefinition AttributeId="hasChief" Name="Is headed by"
Archive="false" Type="Reference" >
    <Target TargetId="Person" Name="Persons" />
  </AttributeDefinition>
  <AttributeDefinition AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" Name="Name" Type="Literal" DataType="xsd:string"
Archive="false" MaxCardinality="1" />
  <ObjectType Code="FormalOrganisation" Name="Formal organisation"
Archive="false" >
    <Parent ParentId="Agent" />
    <ApplicableAttribute AttributeId="hasChief" />
```

```
    <ApplicableAttribute AttributeId="http://www.w3.org/2000/01/rdf-
schema#label" />
  </ObjectType>
</DataSchemaCompact>
```

4.4. Get individuals

4.4.1. Get one individual by URI

GetObject – request one individual by URI

Parameters

Code – individual URI.

Sample query

XML format:

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

JSON format:

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

Response

Items – packet with properties of the individual, described in the next section.

4.4.2. Get a set of individuals with filters and sorting

GetObjectsGroup – request for a set of individuals of some class matching filter conditions.

Parameters

The request has two formats. In compact format the request consists of one GetObjectsGroup tag, the Code attribute of which specifies the URI of class all of which individuals has to be returned.

In full format, the GetObjectsGroup tag includes several nested ObjectType tags, which allows you to select instances that belong to more than one class (at least one of those listed). The FilterGroup and Filter tags allow you to set filtering conditions. Their syntax is described below.

GetObjectsGroup tag

Request root tag.

Attributes:

Code – URI of a class which individuals has to be returned (in the compact request format).

All the other tags and attributes are used only in the full request format.

Root tag attributes:

ObjectTypeGroupOperation – can be "and" or "or". It is used to process conditions of belonging of individuals to the classes specified in the ObjectType tags. The attribute is optional; default value is "or".

CombineGroups – optional attribute, can be "and" or "or". It is used to group several filter criteria, specified by the FilterGroup tags. Default value is "and".

CombineGeometry – optional attribute, can be "and" or "or". It is used to group several selection criteria by coordinates specified by Geometry tags. Default value is "or".

ReturnCount – optional attribute, takes values 0 and 1, default is 0. If set to 1, the number of individuals will be returned in the Response packet, instead of objects themselves.

WithoutSubClasses – optional attribute with values 0 and 1, default is 0. If set to 1, then only objects belonging directly to the specified classes will be returned. If the WithoutSubClasses attribute is 0 or is not set, then the objects of all subclasses of the classes passed in the ObjectType tags and / attribute Code of the GetObjectsGroup tag also will be returned.

ReturnCodeOnly – optional attribute, takes values 0 and 1, default is 0. If ReturnCodeOnly is 1, instead of full information about the object, only URIs of objects that satisfy the filter conditions will be returned in Response: for each object, the Item tag with a single Code attribute is returned and without nested Type and Attribute tags.

Limit – integer, optional attribute, defaults to 1000. Means the number of objects returned.

Offset – integer, optional attribute, defaults to 0. Means the number of objects to be skipped in the search result. The objects matching search criteria are numbered from 0.

ObjectType tag

Allows to define URIs of the classes which individuals has to be returned.

Attributes:

Code – URI of the class which individuals has to be returned.

The request may contain several ObjectType tags with different Code values. Depending on the value of the ObjectTypeGroupOperation attribute of the GetObjectsGroup tag, either the individuals of at least one of the listed classes are returned (with ObjectTypeGroupOperation = "or", or if the value of the ObjectTypeGroupOperation attribute is not specified), or to all of the listed classes (if the ObjectTypeGroupOperation value is "and"). This takes into account the classes hierarchy: objects belonging to subclasses of the requested classes are also returned (unless the WithoutSubClasses = 1 attribute is passed in the root tag of the request).

FilterGroup tag

A group of filter conditions. The CombineGroups attribute of the root GetObjectsGroup tag defines the logical operator which will combine filter groups, "and" or "or". By default, groups are joined by logical AND.

FilterGroup tag attributes

Operation – a logical operation that combines filter conditions specified within a group. Accepts the values "and" and "or".

Filter tag

Defines one filtering condition.

Filter tag attributes

Attribute – URI of the attribute to which values applies the conditions

Value – a literal value (for literal attributes) to compare the value of the specified attribute if the individuals selected. For the reference attributes, Value must contain the identifier of the target individual object.

Comparison – comparison operation between the value of the "Value" attribute and the value of the individual's attribute:

- Equal,
- Contains,
- More,
- Less,
- MoreOrEqual,
- LessOrEqual,
- NotEqual,
- Exists (the value is set),
- NotExists (the value is not set),
- iEqual (case-insensitive equal).

Variable – optional attribute, contains the name of the variable participating in the condition (for multilevel queries, see below). The use of variables in filters allows one to build rather complex selection conditions that take into account chains of links and impose conditions not only on the properties of the object itself, but on the properties of related objects.

ByName – an optional attribute which can take values 0 and 1, the default value is 0. Only meaningful for reference attributes. If ByName is 1, the readable name of the object is used for comparison with Value instead of identifier.

Geometry tag

Allows you to search for objects by geographic coordinates saved in the GeoJSON format. It is not applicable to all data, but only to those whose storage allows this kind of search. If the storage does not support geosearching, the condition will always be false. With the rest of the conditions, the condition on the coordinates is combined with a logical AND. If several Geometry tags are specified, then they will be combined with each other depending on the value of the CombineGeometry attribute of the head tag GetObjectsGroup - logical AND or OR, by default - OR.

Geometry tag attributes:

Type – a type of geometry object, "point" or "polygon".

Attribute – ontology attribute containing coordinates (must have respectively archigraph:point or archigraph:polygon range)

Coordinates tag

Defines the set of polygon vertices in the search by GeoJSON fields

Coordinates tag attributes

Longitude – a geographical longitude, a number between (-180,180]

Latitude – a geographical latitude, a number between (-90,90]

Item tag

Allows to pass identifiers of the searched individuals. The conditions of the Item tag are combined by logical AND with other conditions: if some of the objects with the passed identifiers do not satisfy the restrictions set using the ObjectType, FilterGroup, Geometry tags and head tag attributes, then these objects will not be returned in the Response.

Item tag attributes

Code – URI of an individual

Sample query using the Item tag: among those specified by employee IDs, return only those who work for Alpha company

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Person"/>
  <FilterGroup Operation="And">
    <Filter Attribute="worksIn" Value="Alpha" Comparison="Equal">
  </FilterGroup>
  <Item Code="JohnDoe" />
  <Item Code="JaneDoe" />
</GetObjectsGroup>
```

Sample query с использованием тега Item JSON format:

```
{"GetObjectsGroup": {
    "ObjectType":[
    {"Code":"Person"}
  ],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"worksIn",
             "Value":"Alpha",
             "Comparison":"Equal"
           }
     ]
   }
    ],
    "Item": [
    {"Code":"JohnDoe"},
    {"Code":"JaneDoe"}
    ]
  }
}
```

Sort tag

Requests the sorting of found objects by the specified attributes in ascending or descending order

Sort tag attributes

AttributeId – mandatory, URI of the attribute to be used for sorting.

Direction – Sort direction - ascending ("ASC") or descending ("DESC"). The attribute is optional, by default it sorts in ascending order of the attribute values ("ASC")

SortByName – an optional attribute that takes the values 0 and 1. Only meaningful when sorting with by the values of pointer attributes. If SortByName is 1, sorting is performed not by attribute values, but by the readable names of the objects that the attribute refers to.

Sample query with sorting - sort events by the name of the city in which they occurred, and within the area - by date in reverse order:

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

Sample query with sorting in JSON format:

```
{"GetObjectsGroup": {
    "ObjectType":[{"Code":"Event"}],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"hasDate",
             "Value":"2018-06-00",
             "Comparison":"More"
           }
     ]
   }
    ],
  "Sort":[
      {
     "AttributeId":"happensInCity",
         "Direction":"ASC",
         "SortByName":"1"
   },
      {
     "AttributeId":"hasDate",
     "Direction":"DESC"
   }
  ]
  }
}
```

FieldSet tag

Allows to get in response only the particular attributes values.

FieldSet tag attributes:

Exclude – takes values 0 and 1, default is 0. If the value is 1, then the specified attributes are excluded from the list of object attributes in the response. If the attribute is 0 or not specified, then only the values of the specified attributes will be returned for objects.

Field tag – nested in FieldSet

Description of the attributes that must be either returned or excluded from response, depending on the value of the Exclude parameter.

Field tag parameters:

AttributeId – attribute identifier

Variable – optional - the identifier of the variable from which the return value is taken. Used for queries with subqueries.

SetVariable – optional, identifier of the variable in which the attribute value should be put. Used for queries with subqueries.

Subrequest tag – defines subquery

Allows all the same Parameters and tags as described for the GetObjectsGroup tag. In addition, it can have an optional SetVariable attribute – if it is set, then the value of the variable with the specified name will contain identifiers of objects selected by the subquery condition.

Sample query

Compact XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup Code="Company" ></GetObjectsGroup>
```

Compact JSON format:

```
{"GetObjectsGroup":{"Code":"Company"}}
```

Full XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Company"/>
  <ObjectType Code="Organisation"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="hasName" Value="alpha" Comparison="equal">
    <Filter Attribute="hasName" Value="beta" Comparison="contains">
  </FilterGroup>
</GetObjectsGroup>
```

Full JSON format:

```
{"GetObjectsGroup":{
  "Originator":"test",
  "ObjectType":[
    {"Code":"Company"},
    {"Code":"Organisation"}
  ],
```

```
  "FilterGroup":[
    {
      "Operation":"Or",
      "Filter":[
      {"Attribute":"hasName", "Value":"alpha", "Comparison":"equal"},
      {"Attribute":"hasName", "Value":"beta", "Comparison":"equal"}
      ]
    }
  ]
 }
}
```

Sample query with geometry filter

XML format:

```
<GetObjectsGroup>
  <ObjectType Code="Event"/>
  <FilterGroup Operation="And">
    <Filter Attribute="hasDate" Value="2018-06-00" Comparison="More" />
  </FilterGroup>
   <Geometry Type="Polygon" Attribute="Coordinates">
    <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>
```

JSON format:

```
{"GetObjectsGroup": {
    "ObjectType":[{"Code":"Event"}],
    "FilterGroup":[
   {
     "Operation":"And",
     "Filter":[
       {
             "Attribute":"hasDate",
             "Value":"2018-06-00",
             "Comparison":"More"
         }
     ]
   }
    ],
    "Geometry": [
    {
      "Type":"Polygon",
      "Attribute":"Coordinates",
          "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"}
          ]
    }
    ]
  }
}
```

Sample query with FieldSet

Return only the "Territory" attribute of the "Park" class individuals.

XML format:

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

JSON format:

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

Sample query with subquery: return all the employees of the active companies with the last name "Doe". For each person return his or her name and the name of the company:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObjectsGroup>
  <ObjectType Code="Person"/>
  <FilterGroup Operation="And">
    <Filter Attribute="hasName" Value="Doe " Comparison="Contains">
    <Filter Attribute="worksIn" Variable="?x" Comparison="Equal">
  </FilterGroup>
  <Subrequest SetVariable="?x">
    <ObjectType Code="Company"/>
    <FilterGroup Operation="And">
     <Filter Attribute="hasStatus" Value="Active" Comparison="Equal" />
    </FilterGroup>
    <FieldSet>
      <Field AttributeId="hasFullName" SetVariable="?y" />
   </FieldSet>
  </Subrequest>
  <FieldSet>
    <Field AttributeId="hasName" />
    <Field AttributeId="worksIn" Variable="?y" />
  </FieldSet>
  <Sort AttributeId="hasName" Direction="ASC" />
</GetObjectsGroup>
```

Response

Items package

Items package contains information of the ontology entities (by the way, classes and attributes may be queries as well as individuals). This packet is returned in response to GetObject, GetObjectsGroup, GetObjectHistory queries.

A package consists of one or more Item elements, each of which represents information about one object. Within one Items package information about several interrelated entities of different types, or about several entities of the same type may be returned.

Each object (entity) is desribed by the following parameters:

- **Identifier** – an object URI in the form of http://some-domain.com/prefix/object. The first part of the identifier (before "object" word) is the prefix. The default prefix of an ontology may be omitted when referencing its entities. URIs of the objects with non-default prefixes are returned in full. The last part of the URI, "object" in the example above, is the unique part of an object identifier.
- **Name** – a standard property applicable to all the entities. Keeps a readable name of an element.
- A set of **types** – a list of classes to which an object belongs. Each object can belong to any number of classes. Class membership is recursive, that is, for ArchiGraph the fact that an object belongs to a certain class of the hierarchy means that it is simultaneously a member of all upper classes.
- A set of **attributes** – a list of property values for this object. For literal properties, an immediate value is specified, for reference properties, a unique identifier of the object to which the property refers. In the semantic model, each property of each object can have as many values as the information model allows.

Example of the Items package representing its overall structure in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="JohnDoe" Name="John Doe">
   <Type TypeId="Person" Name="Persons" />
   <Type TypeId="Employee" Name="Employee" />
   <Attribute Type="Literal" AttributeId="hasBirthDate" Value="1979-01-01"/>
   <Attribute Type="Reference" AttributeId="worksIn" Value="AlphaLLC"/>
  </Item>
</Items>
```

The same packet in JSON format:

```
{"Items": {
  "Destination":"test",
  "Item": [
    {
    "Code":"JohnDoe",
    "Name":"John Doe",
    "Type": [
      {"TypeId":"Person", "Name":"Persons"},
      {"TypeId":"Employee", "Name":"Employee"}
    ],
    "Attribute": [
      {
        "Type":"Literal",
        "AttributeId":"hasBirthDate",
         "Value":"1979-01-01"
      },
      {
        "Type":"Reference",
        "AttributeId":"worksIn",
        "Value":"AlphaLLC"
```

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

Item tag

Contains information about a specific object. The Item tag can either be returned as part of the Items, SubscriptionItems, SubscriptionDeleteItems packages, or received as part of the incoming UpdateObject package containing the request to edit / create the object. Below is a description of the attributes of the Item tag, specifying when it can be specified.

Item tag attributes

Code – entity URI

Name – entity readable name

LocalCode – the temporary object code used when the Item tag is submitted as part of an UpdateObject packet containing a request to create an object. The attribute value is the internal code of the object in the system or component in which it was created. In this case the Code attribute is passed empty. In the response package OperationResults, the system returns the permanent unique Code (URI) assigned to the object, as well as the correspondence between it and the LocalCode.

OperationId – a unique identifier of the operation to create / modify an object. Used when passing the Item tag as part of an UpdateObject package. Generated by the sending application, used to get information about the operation result.

DomainIntersection – optional attribute. It can be used only when updating property with Item tag within UpdateObject request. DomainIntersection = true is specified if the domain of the property is the intersection of a set of classes. By default, if a property has several classes in the "domain" attribute, the scope of the property will be the union of these classes; DomainIntersection is not specified in the Item tag in this case.

RangeIntersection – optional attribute, also only applies to updating an ObjectProperty entity. RangeIntersection = true set if the scope of the reference property is class intersection; otherwise, if the range is a union of classes, RangeIntersection is not specified.

IgnoreTypes – an optional attribute which is used when passing the Item tag as part of the UpdateObject package, takes values 0 or 1, the default value is 0. When the attribute is set to 1, the object's class in the Type tags inside the Item tag can be omitted; if the Type tags are set, then their values will be ignored during requests processing – the object's existing classes set will be preserved. It makes sense only for the case of object editing. Request to create an object with IgnoreTypes = 1 will return an error.

AddTypes – an optional attribute which is used when passing the Item tag as part of the UpdateObject package, takes values 0 or 1, the default value is 0. When AddTypes = 1, the classes passed in the Type tags will be added to the object's classes (if they were previously set). With AddTypes = 0 and without specifying IgnoreTypes = 1 (by default) the set of classes to which the object belongs will be replaced with the set of classes passed in the Type tags.

CreateIfNotExists – optional attribute, used when passing a tag as part of an UpdateObject request. If the CreateIfNotExists = 1 attribute value is passed for the Item tag and the Code attribute is set, but the there is no entity with the specified identifier in the platform storages, then a new object will be created with an URI equal to the value of the Code attribute. The default CreateIfNotExists value is 0 – then if an object with an identifier from the Code attribute is absent in the system, an error message will be returned.

FullUpdate – optional attribute, used when passing a tag as part of an UpdateObject request. Takes the values 0 and 1, the default value is 0. If the FullUpdate attribute is not specified or it is equal to 0, then changes will only affect those attributes that were indicated in the request. Otherwise, the object will be updated completely: all existing values of the attributes not listed in the request will be deleted, with the exception of the system attributes SourceSystem and LocalCode.

Date – date and time type – returned if Item contains a Response to a request to get the state of an object for a specified date (GetObject or GetObjectsGroup with an additional Date parameter).

Type tag – nested into Item

Contains information about which classes this object belongs to. Can be specified multiple times for one object.

Type tag attributes

TypeId – an identifier of the class to which the object belongs.

Name – readable name of the class, meaningful only in the packet from the platform to the client. Ignored in UpdateObject package.

Operation tag – nested into Item

It is returned if a response is received for a GetObjectHistory request and contains information about the history of changing the object's class ownership.

Operation tag attributes:

Date – date and time type – the moment when the attribute value was set

System – a code of the system / component which issued update request

OperationId – identifier of the operation which has updated the attribute value (if it was specified in the update request)

Nested tags for Operation tag: Set (there may be one, several or none such tags – in case if attribute value was cleared). The only attribute of the Set tag, Value, contains the value set.

Attribute tag – nested in Item

Contains information about the attribute value of an object. May occur multiple times for the same attribute if the data model allows it.

Attribute tag attributes

Type – attribute value type: Literal for the values of XSD types, Reference for the URIes of another objects, and LocalCodeReference if a reference is made to another object by its LocalCode. The latter option is used only when the Item tag is passed as part of the UpdateObject package. In this case, the package may contain a request to create several entities at once, related to each other. The LocalCodeReference value is used to indicate that the Value refers to another object created in the same request by its temporary code.

AttributeId – attribute URI

Value – attribute value

Name – the readable name of the object which identifier is contained in the attribute value. Returned for reference type attributes only. Will be ignored if the value of the Name attribute is passed as part of the UpdateObject request.

Ignore – used when passing the Item tag as part of the UpdateObject package, and indicates that the system should not change the value of this attribute.

Empty – used when passing the Item tag as part of an UpdateObject package, specifying that the attribute value should be cleared.

ExistingOnly – used when passing the Item tag as part of an UpdateObject package, indicates that the attribute value is changed only if it has already been set previously.

AddValue – used when passing a tag as part of an UpdateObject package, indicates that if the value or values for the attribute have already been set, then the new value does not overwrite the existing ones, but is added to them. Makes sense for multi-valued attributes.

4.5. Update ontology entities

The Originator attribute must be present in every update request.

4.5.1. Create or change entity

UpdateObject – request to create or update an entity or the group of entities defined by their URIs.

`Parameters`

The package contains above described Item tags in the context of the Items package, which contains information about an entity. A request to change several entities at once can be sent in one package. Each entity has its own Item tag. The Item tag within the request for object update must contain the value of the OperationId attribute - this identifier will be specified in the Response and allows you to associate a request to edit a specific entity with the result of execution. The described requests allow creating or updating TBox content (classes and properties) as well as ABox individuals.

`Sample query`

Entity update sample query, XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Person_1" OperationId="0000041">
```

```
    <Type TypeId="Person" />
    <Attribute Type="Literal" AttributeId="http://www.w3.org/2000/01/rdf-
schema#comment" Value="VIP" />
  </Item>
</UpdateObject>
```

Entity update, JSON format:

```
{"UpdateObject": {
 "Originator":"test",
 "OperationId":"000004",
 "Item": [
    {
   "Code":"Person_1",
   "OperationId":"0000041",
     "Type": [
     {"TypeId":"Person"}
   ],
   "Attribute": [
    {
      "Type":"Literal",
      "AttributeId":"http://www.w3.org/2000/01/rdf-schema#comment",
          "Value":"VIP"
          }
    ]
    }
  ]
 }
}
```

Entity create sample query, XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" OperationId="000004">
  <Item LocalCode="1298" OperationId="0000041">
    <Type TypeId="Person" />
    <Attribute Type="Literal" AttributeId="FullName" Value="John Doe" />
  </Item>
</UpdateObject>
```

When editing objects, only the values of the attributes passed in Attribute tags will be changed by default. Object attributes that are omitted in the request will retain their values. If you want to leave only the passed attributes values for the object and remove all others, then you must specify the FullUpdate = 1 flag in the Item tag.

To clear the values of a particular attribute, use the Empty flag in the Attribute tag. Sample query which deletes all values of the Address attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Organisation_1"   OperationId="0000041">
    <Type TypeId="Organisation" />
    <Attribute Type="Literal" AttributeId="Address" Empty="1" />
  </Item>
</UpdateObject>
```

If the attribute can have multiple values, then editing will delete all the old attribute values and assign the values passed in the request. To add values to the list of previously set values, you must use the AddValue flag for the Attribute tag.

Consider an example of using AddValue. Let the Organisation_1 object have the value of the participatingInProject attribute value equal to Project_1. As a result of the query:

```
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Organisation_1"   OperationId="0000041">
    <Type TypeId="Organisation" />
    <Attribute Type="Reference" AttributeId="participatingInProject"
Value="Project_2"/>
  </Item>
</UpdateObject>
```

the participatingInProject attribute value will be set to Project_2. In contrary, as a result of the query:

```
<UpdateObject Originator="test" OperationId="000004">
  <Item Code="Organisation_1"   OperationId="0000041">
    <Type TypeId="Organisation " />
    <Attribute Type="Reference" AttributeId="participatesInProject"
Value="Project _2" AddValue="1" />
  </Item>
</UpdateObject>
```

the participatingInProject attribute will have two values, Project_1 and Project_2.

The Ignore flag of the Attribute tag means that the attribute does not need to be considered when updating object; Attribute tags with this flag will be skipped when processing the package.

When updating an object, the IgnoreTypes flag for the Item tag can be used. In this case, the nested Type tags can be passed or not passed – anyway, the object will be still belonging to the classes that it belongs. If the IgnoreTypes flag is not specified, then the Type tags are required and the object will be belonging to each of classes from the list passed in the request. When creating a new object the Type attribute is required, and using the IgnoreTypes flag will return an error.

Another flag that affects the handling of object class membership is AddTypes. If AddTypes = 1 for an existing object, the list of classes to which it belongs will be supplemented with the list of classes passed in the request. For the new object, the list of passed classes will be assigned in the same way as without the AddTypes flag. If the IgnoreTypes and AddTypes flags are set to 1 at the same time, the AddTypes flag will be ignored.

Response

OperationResults package – a packet describing operation results.

Nested tags: OperationResult – operation results for a single object in the packet.

OperationResult tag attributes:

OperationId – request identifier, a value copied from the Item tag of the request.

Result – can contain values of *success*, *error* or *proposed*. The *success* value will be returned if object was update successfully, the *error* value – in case of any abnormal case during request execution. If the requesting system or component has the "editing with confirmation" rights to the object being updates, and the update request was successfully registered in the platform, the *proposed* value will be returned in the Result attribute.

Message – the text of the error message in case of failure, or text describing the performed operation in other cases; for example, in a group operation, Message will contain information about the number of changed or deleted objects.

Code – URI of the created or updated object

LocalCode – the object's code defined by the requesting system / component, if it was set in the request.

Response example

XML format

```
<OperationResults Destination="test" >
  <OperationResult Result="success" OperationId="0000041" Code="Person_1"
LocalCode="1297" />
  <OperationResult Result="error" Message="Unknown code Person_007"
OperationId="0000042" Code="Person_007" />
</OperationResults>
```

JSON format:

```
{"OperationResults":  {
 "Originator":"test",
 "OperationId":"000004",
 "OperationResult":   [
    {
    "Result":"success",
    "Message":"Unknown code Person_007",
      "OperationId":"0000041",
    "Code":"Person_1",
    "LocalCode":"1297"
  },
    {
    "Result":"error",
    "Message":"Unknown code Person_007",
      "OperationId":"0000042",
    "Code":"Person_007"
  }
  ]
 }
}
```

In addition, after the successful completion of the operation, ArchiGraph sends a SubscriptionItems packet, which is received by all information systems interested in information about objects of this type. The SubscriptionItems package has exactly the same structure as the Items package described in the previous section. For objects in the SubscriptionItems package, the complete set of their attributes is passed, not just the modified ones.

4.5.2. Delete entity

DeleteObject – request to delete entity by its URI.

Parameters

Code – entity URI

Response

OperationResults package – a package containing information about the operation results. In addition, after the successful completion of the operation, ArchiGraph sends the SubscriptionDeleteItems package to information systems subscribed to changes of objects of this type. The structure of the package is identical to the Items package, it contains an Item tag describing the entity to be deleted.

Sample query

XML format:

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

JSON format:

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

4.5.3. Update a set of objects

UpdateObjectsGroup – request to update group of objects

Parameters

The request has two formats – full and short, similar to the GetObjectsGroup request formats.

In shorthand format, a request consists of an *UpdateObjectsGroup* tag and one or more Attribute tags attached to it, which characterize the object attributes that will be changed. The Attribute tag is described in the structure of the Items package. For the UpdateObjectsGroup tag, an additional Code attribute is set - the identifier of the model class. Executing a short format query will update all objects belonging to the specified class.

In full format, unlike the short one, the UpdateObjectsGroup tag does not have a Code attribute, but instead includes several nested ObjectType, FilterGroup, Item, Geometry tags that allow you to set a condition for selecting those objects that will be changed. The syntax for the ObjectType, FilterGroup, Item, Geometry tags is given in the GetObjectsGroup package structure.

Response

OperationResults – package with information about the operation result. The Response text will return information about the number of objects changed. ArchiGraph also sends out SubscriptionItems packages with all objects changed as a result of a query. All the subscribed information systems / components that have permission to access these objects will get SubscriptionItems package.

Example

Compact format:

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

Full format:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObjectsGroup Endpoint="demo" Originator="test"
OperationId="0945ab4fr4">
  <ObjectType Code="Company"/>
  <ObjectType Code="Organisation"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="FullName" Value="Alpha LLC" Comparison="Equal">
    <Filter Attribute="FullName" Value="Beta LLC" Comparison="Equal">
  </FilterGroup>
  <Attribute Type="Literal" AttributeId="isActive" Value="false" />
</UpdateObjectsGroup>
```

4.5.4. Delete group of objects

DeleteObjectsGroup – request to delete a group of objects.

Parameters

This query has the compact and full syntaxes similar to the GetObjectsGroup request syntax.

In the compact syntax the request has one DeleteObjectsGroup tag with the Code attribute containing URI of the class all the individuals of which has to be deleted.

In the full syntax DeleteObjectsGroup tag can contain nested ObjectType, FilterGroup, Item, Geometry tags allowing to define select conditions. Only the individuals matching these conditions will be deleted. The ObjectType, FilterGroup, Item, Geometry tags syntax if described above in the GetObjectsGroup request description.

Response

OperationResults – a packet describing the results of operation, including number of deleted objects. The platform also sends SubscriptionDeleteItems packets containing identifiers of all deleted objects to all the systems/components subscribed to the classes to which these objects were belonging.

Example

Compact syntax:

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

Full syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeleteObjectsGroup Endpoint="demo" Originator="test"
OperationId="0945ab454dcf7" CombineGroups="Or">
  <ObjectType Code="Company"/>
  <ObjectType Code="Organisation"/>
  <FilterGroup Operation="Or">
    <Filter Attribute="hasName" Value="Alpha" Comparison="contains">
    <Filter Attribute="hasName" Value="Beta" Comparison="contains">
  </FilterGroup>
</DeleteObjectsGroup>
```

4.6. Multilingual data

4.6.1. Get the list of supported languages

GetLanguages – get the list of languages supported by the platform. The method has no parameters.

Example

XML format:

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

JSON format:

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

Response

LanguagesList – a packet with the list of supported languages.
The nested *Language* tags are describing supported languages and have the following attributes:
Code – language code according to ISO 639.

Name – readable name of the language.

Default – indicates if this language is the platform's default language, returns false or true.

Response example

XML format:

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

JSON format:

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

4.6.2. Get multilingual data of ontology entities

In the platform settings one of the languages is set as the default language. When requesting objects, if the language is not specified explicitly and there is data in several languages for the object, then only the data in the default language will be returned.

To request data in other languages, use the Lang attribute specified in the root request. The attribute value can be the code of one of the languages supported by the system, or the ALL keyword, which allows you to get data in all the languages specified for the object.

Example

Get object data in English, XML format:

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

JSON format:

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

Get object data in all languages, XML format:

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

Get all the individuals of Person class with the literal properties in English, XML format:

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

In the returned packets the language is indicated in the Lang attribute of the Attribute tag.

Response example in XML format, in which the rdfs:label properties values are set in two languages:

```
<Items Endpoint="demo" Destination="system" Lang="ALL" >
  <Item Code="Person" Name="Persons">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Class"/>
    <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>
```

If the data language matches the platform's default language, the Lang attribute may be omitted. Response example in case if the English is a default language:

```
<Items Endpoint="demo" Destination="system" Lang="ALL" >
  <Item Code="Person" Name="Persons">
    <Type TypeId="http://www.w3.org/2002/07/owl#Class" Name="Class"/>
    <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" />
  </Item>
</Items>
```

Only literal properties of xsd:string datatype may have values on different languages. The values of the properties of other data types will be returned on any request for an object. For reference attributes, the readable name of the referenced object will be returned in the default language if the language is not specified in the request or the Lang = ALL option is selected. If data is requested in a specific language, and a rdfs:label property value in that language is set for the referenced object, it will be returned in the Name attribute of the Item tag. If the referenced object property does not have a readable name in the requested language, then the name in the default language will be returned in the Name attribute of the Item tag.
An example of the packet received when requesting without specifying the Lang attribute, the default language is English:

```
<Item Code="Company_1" Name="Doe and partners">
  <Type TypeId="Company" Name="Companies"/>
  <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Doe and partners"/>
  <Attribute AttributeId="incorporatedAt" Type="Literal" Value="2001-01-02"/>
  <Attribute AttributeId="isChildOf" Type="Reference" Value="Company_2" Name="Doe
Inc."/>
</Item>
```

The example of the packed received in response to the request with Lang=DE:

```
<Item Code="Company_1" Name="Doe und Partner">
  <Type TypeId="Company" Name="Companies"/>
  <Attribute AttributeId="http://www.w3.org/2000/01/rdf-schema#label"
Type="Literal" Value="Doe und Partner" Lang="DE"/>
  <Attribute AttributeId="incorporatedAt" Type="Literal" Value="2001-01-02"/>
  <Attribute AttributeId="isChildOf" Type="Reference" Value="Company_2" Name="Doe
Gmbh"/>
</Item>
```

4.6.3. Update multilingual data

When updating objects, the language of every property value is specified in the Lang attribute of the Attribute tag. The language can also be specified in the Lang attribute of the Item tag, in this case it applies to all property values passed in the request.

Example

Set the readable name of an object in several languages, XML format:

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

Set the values of several attributes in English, XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Endpoint="demo" Originator="system"
  <Item Code="Person" OperationId="1234" Lang="EN">
    <Type TypeId="owl:Class" />
```

```
    <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="A class for representing the Persons" />
 </Item>
</UpdateObject>
```

Only literal properties of xsd:string datatype may have values in different languages. If the data language coincides with the default language, then it can be omitted. If a language version is specified for a property whose type does not allow data in different languages (numeric, date, etc.), then an error will be returned.

Example of an error returned when trying to assign a language version to a boolean property:

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateObject Endpoint="demo" Originator="test" OperationId="000004">
  <Item  Code="Person" 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>
```

Response:

```
<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="Person"/>
</OperationResults>
```

4.7. Subscriptions

4.7.1. Subscribe on objects updates

UpdateSubscription – subscribe a system / component for receiving messages on updates of some class(-es) individuals or the class model.

Parameters

Nested tag: *Subscribe*

Subscribe tag attributes:

Format – the format of received packets containing information about updated objects, takes the value "XML" or "JSON". It is a mandatory attribute when creating a new subscription of an information system to a given class. May not be specified when changing the parameters of an existing subscription.

OperationId – optional string. The value that will be passed in the OperationId attribute of the SubscriptionItems and SubscriptionDeleteItems packages. Serves to distinguish between packages received under different subscriptions.

Delayed – optional attribute, takes values 0 (used by default) or 1, indicates whether the subscription is delayed. If the value is 0, then the package with the changed objects will be

sent immediately after the changes are made. If there is no need for an immediate response, you can use delayed sending, which reduces the load on the system and does not slow down the processing speed of requests on the ArchiGraph side.

Objects – optional attribute, takes the values 0 (used by default) or 1. Indicates whether to receive information about changes in individuals belonging to the classes specified in the subscription.

Model – optional attribute, takes the values 0 (used by default) or 1. Indicates whether to receive information about changes of the class itself or the attributes applicable to its individuals.

Active – optional, defaults to 1. Indicates whether the subscription is active (value 1) or inactive (value 0).

Exclude – optional parameter, takes values 0 and 1, defaults to 0. Allows to exclude a subclass from subscription. When the messages are sent by subscription the inheritance of classes is taken into account: if Class2 is a subclass of Class1, and the system has subscribed to receive changes to Class1 objects, then the system will also receive changes to Class2 objects. If you do not need to get Class2 objects, you must additionally set a subscription to Class2 with Exclude = 1.

Host, **Port**, **Login**, **Password**, **Queue** – queue details for receiving packages about changing entities. If none of these attributes is passed, but an active subscription is already registered for the system, then the details of the existing subscription will be used.

Broker – optional attribute, message broker. Supported values for "RabbitMQ" and "ApacheKafka". If not specified, then the value specified by the MDM settings as the default broker value will be used.

Originator, Token – optional string attributes. If set, they will be passed in the root tag of the package received by subscription.

ObjectType tag – nested in Subscribe tag.

Lists model classes to subscribe to changes to objects or properties.

ObjectType tag attributes:

Code – class URI

Sample query

Request to add to existing subscriptions a subscription to receive information about objects of the Contractor and Employee classes:

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

Response

OperationResults package – information on the operation results.

To subscribe to receive information about changes in all model classes, a service word __root__ is specified as a class in the ObjectType tag.

Sample query

Subscribe on updates of the whole model:

```
<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 – discover if the system / component is subscribed to the updates of individuals and model of some class and get subscription properties. If class is not given in the request parameters, all the classes will be returned.

Parameters

ObjectType tag

ObjectType tag attributes:

Code – class identifier

Sample query

XML format:

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

JSON format:

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

Response

Subscribes package

Nested tage

Subscribe – describes an individual subscription. If the system / component has no registered subscriptions for the class specified in the request, then the Subscribes tag will not contain any nested tags.

Subscribe tag attributes:

Active – shows if the subscription is active (1) or inactive (0);

Format – subscription packets format, XML or JSON;

OperationId – the value of OperationId attribute of SubscriptionItems and SubscriptionDeleteItems packets;

Delayed – shows if the subscription is delayed (1) or not (0);

Objects – shows if subscribed to the individuals changes (1) or not (0);

Model – shows if subscribed to the class model changes (1) or not (0);

Host, **Port**, **Login**, **Password**, **Queue**, **Broker** – queue requisites for the subscription messaes.

Nested tags: *ObjectType*

ObjectType tag attributes:

Code – class URI
Name – class readable name

Response example

XML format:

```
<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="Organisation" Name="Organisations" />
    <ObjectType Code="Person" Name="Persons" />
  </Subscribe>
</Subscribes>
```

Subscriptions consider the class hierarchy. If Class2 is a subclass of Class1, then objects and properties of class Class2 will be applied to all subscriptions made to class Class1. Query for the status of Class2 subscriptions will return a description of the actual subscription.

Sample query for subclass:

```
<GetSubscription Endpoint="demo" Originator="test">
  <ObjectType Code="Class2" />
</GetSubscription>
```

Response on the subclass subscription query:

```
<Subscribes Endpoint="demo" Destination="test">
  <Subscribe Format="json" [ … ]>
    <ObjectType Code="Class1" Name="Class 1" />
  </Subscribe>
</Subscribes>
```

4.7.3. Cancel subscription

DeleteSubscription – cancel system / component subscription for receiving updates on the model and individuals of the listed classes.

Nested tags: *ObjectType*

Атрибуты ObjectType:

Code – class URI, mandatory

Optional attributes:

Format – if set to 1, only removes subscriptions that return data in the specified JSON or XML format. If not specified, all subscription formats are removed.

Delayed – allows you to delete only immediate (if set to 0) or delayed (if set to 1) subscriptions. If not set, all subscription types are deleted.

Objects – if set to 1, then subscriptions on the updates of the objects of the specified class will be cancelled.

Model – if set to 1, then subscriptions on the updates of the class itself and the attributes applicable to it will be cancelled.

If Objects nor Model are not set to 1, then all kinds of subscriptions will be deleted.

Sample query

XML format:

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

Response

OperationResults packet – information on the operation result.

5. Data and model temporality

5.1. Model temporality

5.1.1. Create virtual endpoint

CreateEndpoint – a request to create a virtual endpoint containing model (TBox) state actual for a given date.

Parameters

Date – a moment in the past for which the model will be restored.

Sample query

XML format:

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

Sample query состояния модели на 01.01.2019:

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

Response

OperationResults packet – information on the operation result. If success, the Code attribute will contain an identifier of the created endpoint. This identifier can be further used as a value

of Endpoint attribute in the requests for the data model (GetDataSchema, GetDataSchemaCompact).

Response example

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

5.1.2. Delete virtual endpoint

DeleteEndpoint – delete virtual endpoint created by CreateEndpoint request.

Parameters

Code – virtual endpoint code

Sample query

XML format:

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

Response

OperationResults packet – information on the operation result.

Response example

XML format:

```
<OperationResults Destination="test" >
  <OperationResult Result="error" Message="demo_20190101000000_24234sad34
endpont not found" Code="demo_20190101000000_24234sad34" />
</OperationResults>
```

5.2. Data history

5.2.1. Get history of an individual

GetObjectHistory – a request for the history of the individual's properties and class membership

Parameters

Code – URI of the individual which history has to be returned.

Sample query

XML format:

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

JSON format:

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

Response

Items packet – the list of the properties and class membership of an individual.

Response example

XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="JohnDoe" Name="John Doe">
   <Type>
    <Operation Date="2019-01-02T16:20:00" System="test"
OperationId="918273645"/>
     <Set Value="Person"/>
     <Set Value="Employee"/>
    </Operation>
    <Operation Date="2019-01-01T00:00:00" System="test">
     <Set Value="Person"/>
    </Operation>
   </Type>
   <Attribute Type="Literal" AttributeId="rdfs:label">
    <Operation Date="2020-01-01T00:00:00" System="test">
     <Set Value="John Doe/>
    </Operation>
    <Operation Date="2019-01-01T00:00:00" System="test">
     <Set Value="Doe, John/>
    </Operation>
   </Attribute>
   <Attribute Type="Literal" AttributeId="hasBirthDate">
    <Operation Date="2019-01-01T00:00:00" System="test">
     <Set Value="1979-01-01"/>
    </Operation>
   </Attribute>
   <Attribute Type="Reference" AttributeId="worksIn">
    <Operation Date="2019-01-03T00:00:00" System="test">
     <Set Value="AlhpaLLC"/>
    </Operation>
    <Operation Date="2019-01-02T16:20:00" System="test"
OperationId="918273645">
     <Set Value="BetaLLC"/>
    </Operation>
   </Attribute>
  </Item>
</Items>
```

Response example JSON format:

```
{"Items": {
  "Destination":"test",
  "Item": [
    {
    "Code": "JohnDoe",
    "Name": "John Doe",
    "Type": [
          {
              "Operation":[
                 {
                   "Date":"2019-01-02 00:00:00",
                   "System":"test",
                   "Operationid":"918273645",
                   "Set":[{"Value":"Person"},{"Value":"Employee"}]
                 },
                 {
                   "Date":"2019-01-01 00:00:00",
```

```
                    "System":"test",
                    "Set":[{"Value":"Person"}]
                  }
                ]
            }
        ],
        "Attribute": [
            {
                "Type":"Literal",
                "AttributeId":"rdfs:label",
                  "Operation": [
                  {
                        "Date":"2020-01-01 00:00:00",
                        "System":"test",
                        "Set":[{"Value":"John Doe"}]
                    },
                  {
                        "Date":"2019-01-01 00:00:00",
                        "System":"test",
                        "Set":[{"Value":"Doe, John"}]
                    }
                ]
            },
              {
                "Type":"Literal",
                "AttributeId":"birthDate",
                  "Operation": [
                  {
                        "Date":"2019-01-01 00:00:00",
                        "System":"test",
                        "Set":[{"Value":"1979-01-01"}]
                    }
                ]
              },
              {
                "Type":"Reference",
                "AttributeId":"worksIn",
                  "Operation": [
                  {
                        "Date":"2019-01-03T00:00:00",
                        "System":"test",
                        "Set":[{"Value":"BetaLLC"}]
                    },
                  {
                        "Date":"2019-01-02T00:00:00",
                        "System":"test",
                        "Operationid":"918273645",
                        "Set":[{"Value":"AlphaLLC"}]
                    }
                ]
              }
        ]
      }
      ]
    }
}
```


5.2.2. Get individual state for a given date

GetObject and GetObjectsGroup requests can be executed by passing an additional Date parameter (date and time type) to get the state of objects on a given date. In this case, the GetObjectsGroup request can be executed only for storages with native support for 4D

temporality (see Storages section), and GetObject – both for them and for storages for which the history of values is kept by means of ArchiGraph. In case the object storage does not allow getting its state in the past, InvalidPackage will be returned with an error message.

Sample query

XML format:

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

Response

Items packet – a description of an individual. The value of Date parameter will be returned in Item tag.

Response example

XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Items Destination="test">
  <Item Code="JohnDoe" Name="John Doe" Date="2019-01-01 00:00:00">
   <Type TypeId="Персона" Name="Person" />
   <Attribute Type="Literal" AttributeId="rdfs:label" Value="John Doe" />
   <Attribute Type="Literal" AttributeId="birthDate" Value="1979-01-01" />
  </Item>
</Items>
```

5.2.3. History storage initialization

If the initial filling of the model and data storages did not happen using the ArchiGraph platform API requests, initialization of the history storage is required. Two service API methods InitModelHistory and InitObjectsHistory are provided for this.

InitModelHistory – method for the history of data model changes initialization. Its only and mandatory Date parameter has the date and time datatype. It marks the date to which the initial data state will be associated.

Sample query

XML format:

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

Sample query JSON format:

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

OperationResults packet will be returned in response.

When the method is called again with the same date, the existing records about the values of the model elements for the specified date will be deleted and filled again.

InitObjectsHistory – method of initializing the history of changes of objects belonging to the specified classes.

Parameters

Date – date and time that will mark the first history of objects. Mandatory attribute.

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.

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

Sample query

XML format:

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

JSON format:

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

Response

OperationResults packet

6. GraphQL API

An interface for testing GraphQL queries is available at /graphql path in any platform installation. On the left there are links to Example, when you click on each of them, the request area and its parameters in the middle of the page are filled. After clicking the Run button, the Platform Response is displayed on the right side of the page.

You can also execute a GraphQL query from the side of software components at /graphql, passing in POST the values of query parameters, variables, etc. in accordance with the rules for generating GraphQL POST-запросов.

Endpoint and Originator values must be passed in request parameters:

```
{
        "params": {
                "originator": "test",
                "endpoint": "demo"
        }
}
```

6.1. Data retrieve request (query)

JSON query query consists of three fields, of which query and variables are required:

```
{query:"...", variables:{...}, operationName:"..."}.
```

The **query** field contains the query string, and the **variables** field contains the variables used in it. In the **params** variable, you need to list the variables of the request parameters, so you cannot use this variable in the request itself.

GraphQL-Response also comes in JSON format, in which fields with data returned by the request and errors can be transmitted:

```
{data:{...}, errors:[...]}
```

The **data** field with the result will come in the same structure as the **query**.

Here are some examples of queries.

1) Request for an object (an instance of the Company class) by identifier – the Code attribute is URI in ArchiGraph, and return of the values of two attributes: http://www.w3.org/2000/01/rdf-schema#label and hasVATnumber

```
{
  "query":"query {
        Company (id: "Organisation_25134855") {
                http://www.w3.org/2000/01/rdf-schema#label
                hasVATnumber
        }
   }",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

Response:

```
{
  "data":{
     "Company":[
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"Alpha LLC",
"hasVATnumber":"612636222"
}
]
  }
}
```

2) Request individuals filtered by the hasStatus attribute:

```
{
  "query":"query {
    Company (hasStatus: "Status_52370757"){
            http://www.w3.org/2000/01/rdf-schema#label
            hasVATnumber
     }
   }",
  "variables": {
```

```
      "params": {
        "originator": "test"
}
   }
}
```

Response

```
{"data":{
"Company":[
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"Alpha LLC",
"hasVATnumber":"612636222"
},
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"Beta LLC"
},
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"Gamma LLC"
}
 ]
  }
}
```

3) Request object by URI with a subquery: get the object with the Organisation_57122638 identifier and the object that it references by participatesInProject attribute.

```
{
  "query":"query {
            Company (id: "Organisation_57122638") {
                    http://www.w3.org/2000/01/rdf-schema#label
                    hasVATnumber
                    participatesInProject {
                       http://www.w3.org/2000/01/rdf-schema#label
                        hasStatus
                    }
            }
  }",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

Response

```
{"data":{
"Company":[
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"AlphaLLC",
"hasVATnumber":"612636222",
"participatesInProject":[
{
"http:\/\/www.w3.org\/2000\/01\/rdf-schema#label":"Alpha Club
reconstruction",
"hasStatus":"FinishedInTime"
}
]
}
]
  }
}
```

4) Get a set of objects with a subquery: get objects of Company class together with the objects related with them by the participatesInProject attribute.

```
{
  "query":"query {
    Контрагент {
            http://www.w3.org/2000/01/rdf-schema#label
            hasVATnumber
            participatesInProject {
                http://www.w3.org/2000/01/rdf-schema#label
                hasStatus
            }
    }
    }",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

5) Get a set of objects with a filter in a subquery: get individuals of the Company class, which are participating in projects having the Deferred status.

```
{
  "query":"query {
    Company {
            http://www.w3.org/2000/01/rdf-schema#label
            hasVATnumber
           participatesInProject(hasStatus: "Deferred") {
                http://www.w3.org/2000/01/rdf-schema#label
               hasStatus
          }
    }
              }",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

6) Get the individuals with two aliases

```
{
  "query":"query {
        reliableCounterparty: Company(hasStatus: "Status_5350554") {
             http://www.w3.org/2000/01/rdf-schema#label
        }
        nonReliableCounterparty: Company(hasStatus: "Status_52370757") {
               http://www.w3.org/2000/01/rdf-schema#label
        }
}",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

7) Request with the fragment

```
{
```

```
  "query":"query {
        reliableCounterparty: Company(hasStatus: "Status_5350554") {
                ...fieldsSet
    }
        nonReliableCounterparty: Company(hasStatus: "Status_52370757") {
                ... fieldsSet
    }
}

fragment fieldsSet on Company {
       http://www.w3.org/2000/01/rdf-schema#label
       hasVATnumber
       participatesInProject {
               http://www.w3.org/2000/01/rdf-schema#label
               hasStatus
       }
}",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

8) Request with the fragment and default values

```
{
  "query":"query ParticipatingInProjects($status: String = "Deferred",
$counterpartyStatus1: String = "Status_5350554") {
           reliableCounterparty: Company(hasStatus: $counterpartyStatus1) {
                   ...fieldsSet
           }
           nonReliableCounterparty: Company(hasStatus: $status) {
                   ...fieldSet
           }
}

fragment fieldsSet on Company{
           http://www.w3.org/2000/01/rdf-schema#label
           participatesInProject(hasStatus: $status) {
                   http://www.w3.org/2000/01/rdf-schema#label
           }
}",
  "variables": {
      "params": {
        "originator": "test"
}
   }
}
```

6.2. Update requests (mutations)

Updating the data stored in the platform can be performed using a mutation request (analogous to the UpdateObject request in the main platform API).

The logic of constructing a mutation query repeats the structure of the data retrieve query.

Below is a Sample query to add / modify two objects of the Person class. The variables $attributes1 and $attributes2 are passed parameters, which you need to modify for the updated objects. If objects with the specified Code (URI) or LocalCode already exist, then the objects will be changed, otherwise they will be created.

```
{
  "query":"mutation ($attributes1: [Attribute], $attributes2: [Attribute]){
            first:UpdateObject(localCode: "000010002123_3", operationID:
"0000051", typeId: "Person", attributes: $attributes1){
                    id
                     http://trinidata.ru/archigraph-mdm/LocalCode
            }
            second:UpdateObject(localCode: "000010002123_2", operationID:
"0000052", typeId: "Person", attributes: $attributes2){
            id
                     http://trinidata.ru/archigraph-mdm/LocalCode
            }
  }",
  "variables": {
      "params":{
                "Endpoint": "demo",
                "originator": "test",
                "OperationID": 55555
        },
        "attributes1":[
        {
                    "attributeId" : "http://www.w3.org/2000/01/rdf-
schema#label",
                    "type" : "Literal",
                    "value": "John Doe"
                }
        ],
        "attributes2": [
                {
                    "attributeId" : "http://www.w3.org/2000/01/rdf-
schema#label",
                    "type" : "Literal",
                    "value": "Jane Doe"
                }
        ]
  }
}
```

In the response the URI and LocalCode of the created or updated objects will be returned:

```
{"data":{
"first":[
{
"http:\/\/trinidata.ru\/archigraph-mdm\/LocalCode":"000010002123_3",
"id":"Person_76aea552c22d05ecbfc78c13fc6d9da9"
}
],
"second":[
{
"http:\/\/trinidata.ru\/archigraph-mdm\/LocalCode":"000010002123_2",
"id":"Person_f29c5c6559efeb6da6433bcf6231bc66"
}
]
}}
```

7. SPARQL endpoint

ArchiGraph platform supports data reading using the standard SPARQL endpoint interface. Currently, only queries to read data (SELECT) are supported, with some restrictions – without using subqueries and aggregating functions. The interface has experimental status.

To perform the request you should query the URL /mdm/[endpoint]/query?originator=[originator]&query=[query]. You can use Fuseki panel for request testing by setting the ArchiGraph platform API URL in the "SPARQL endpoint" field as shown below:

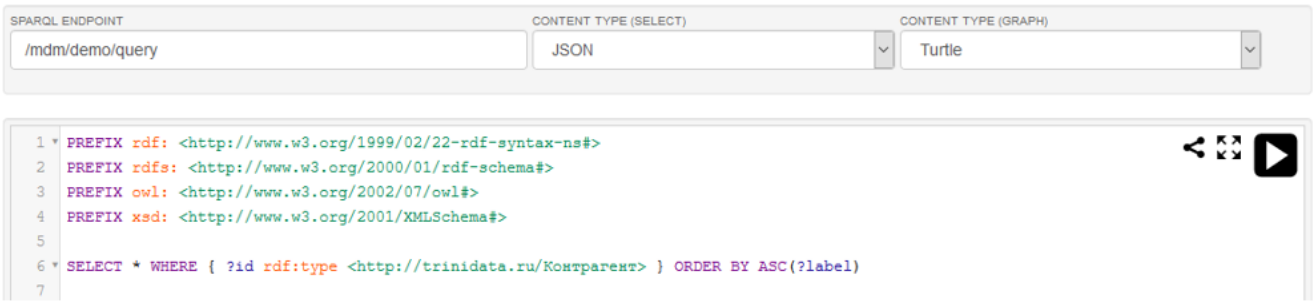

Fig. 2. Testing ArchiGraph platform SPARQL endpoint from Apache Fuseki interface

The response will be returned in JSON form according to the SPARQL speficication.